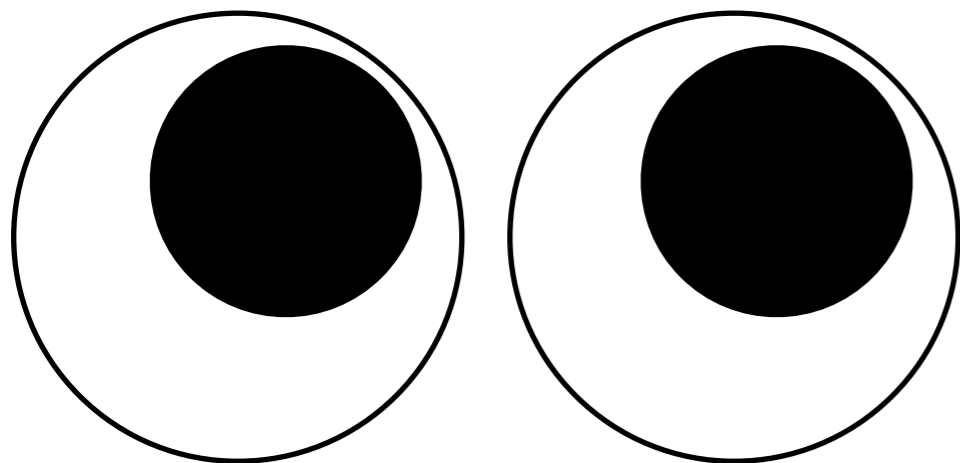


Deep-Pipelined FPGA Implementation of Ellipse Estimation for Eye Tracking

Keisuke Dohi, Yuma Hatanaka, Kazuhiro Negi,
Yuichiro Shibata, Kiyoshi Oguri

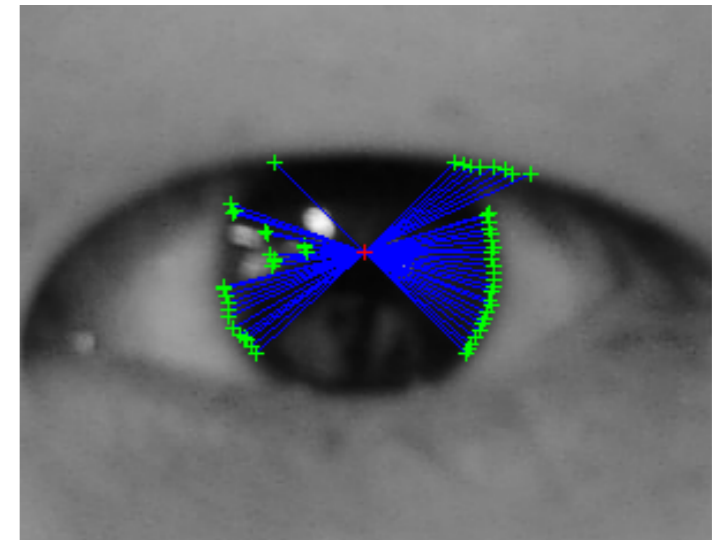
Graduate school of engineering, Nagasaki University, Japan



Background - OpenEyes

Software implementation for eye tracking

- Starburst feature detection
 - Finding feature points
 - Searching in a radial fashion
- RANdom SAmple Consensus (RANSAC) algorithm
 - Estimating an ellipse from set of feature points
 - Iterative computation
 - It needs to solve ellipse equation

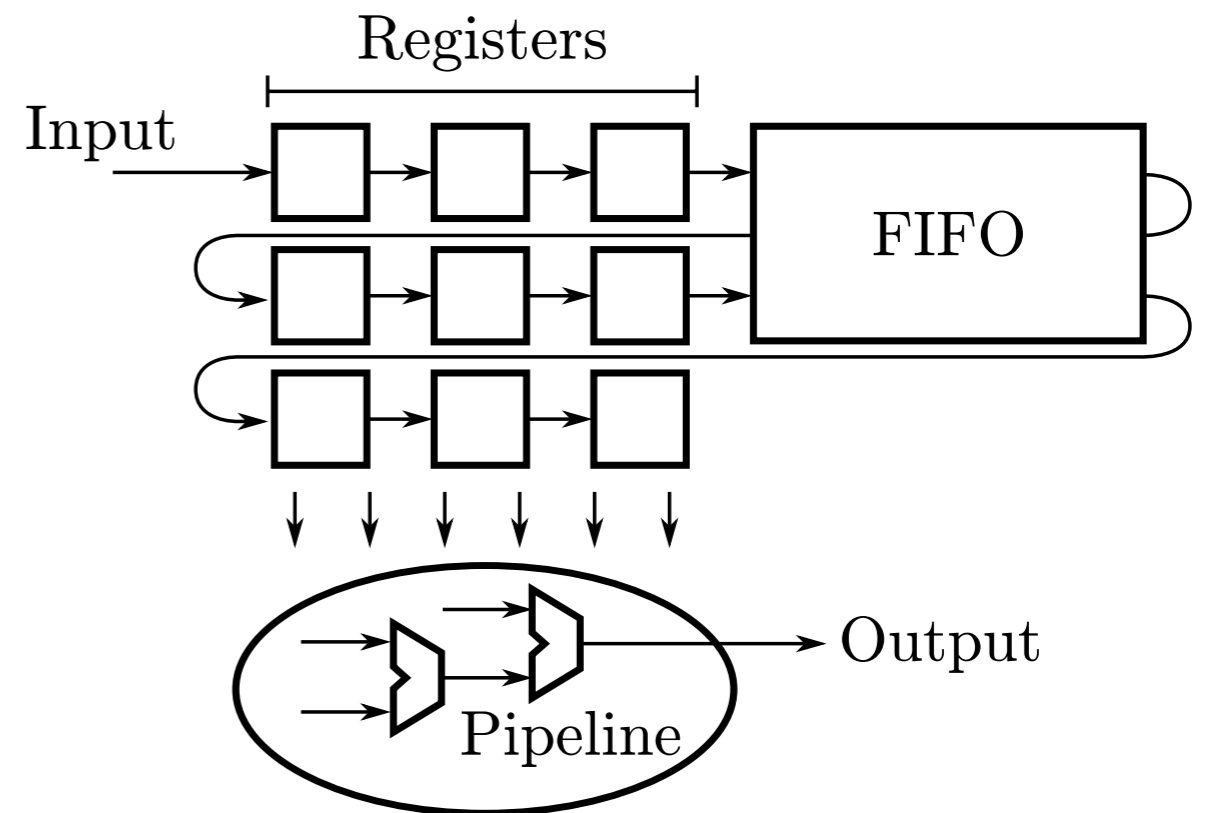


Goal

- **Fast** and **Low-power** FPGA implementation

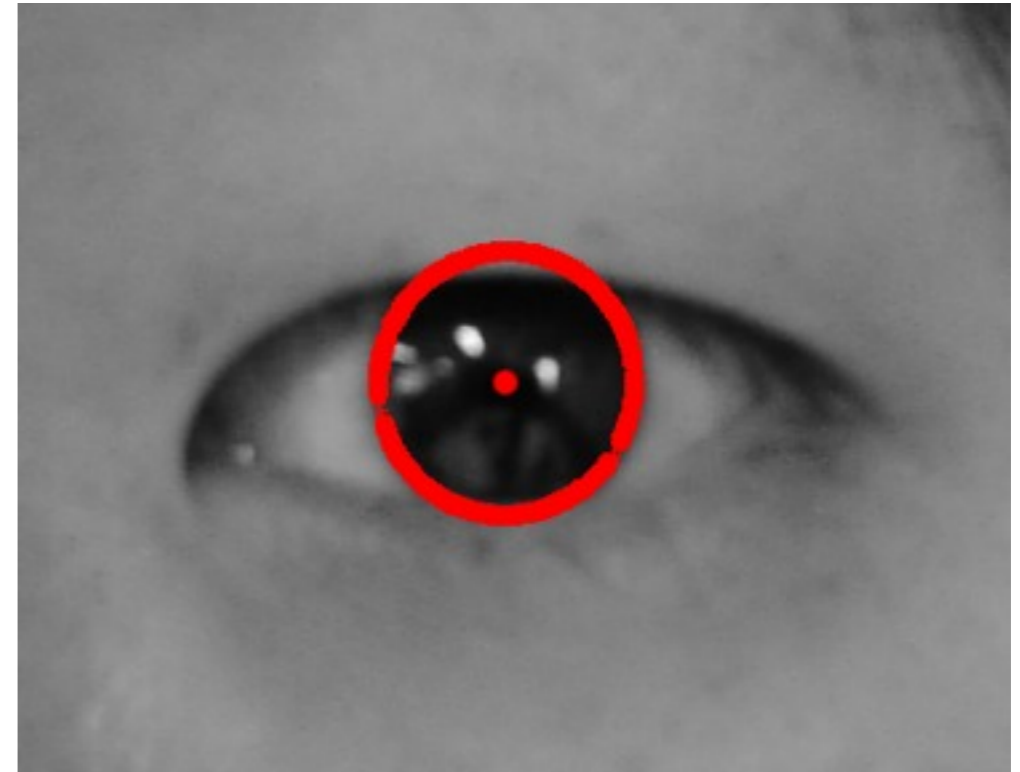
for eye tracking

- Real-time video process
- Stream-oriented process
- Without using any external memory



Overview of our system

1. Camera I/F & Pre-processes
2. Feature detection
3. Random sampling
4. Hypothesis (Ellipse) generation
5. Model verification

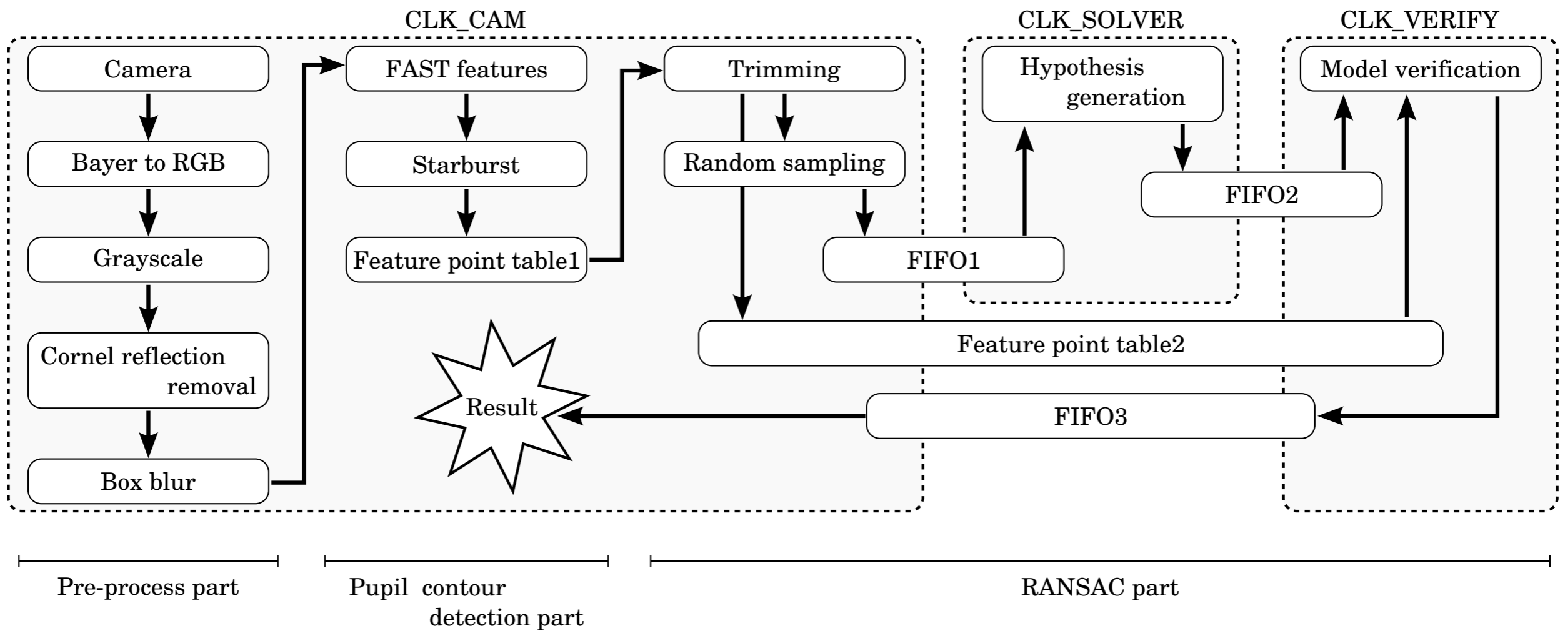


Iterative
computations

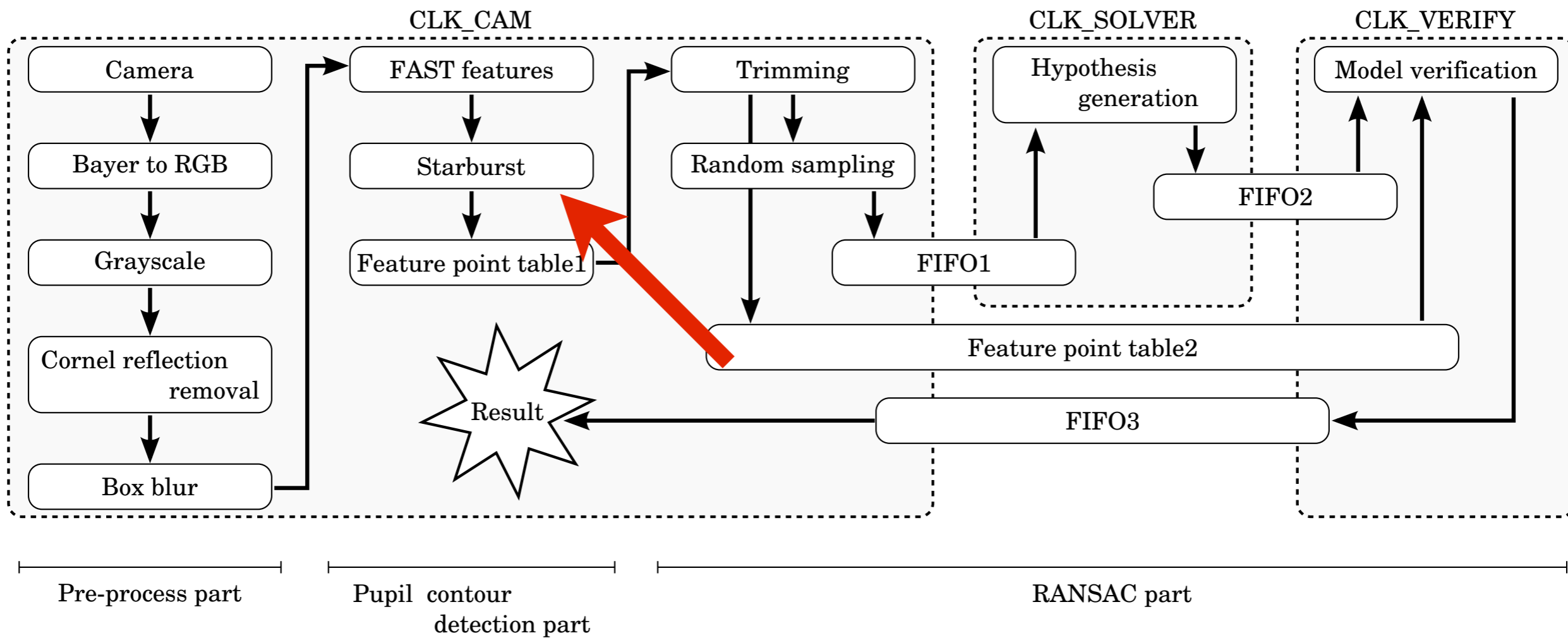
Ellipse :

$$x^2 + Axy + By^2 + Cx + Dy + E = 0$$

Architecture overview

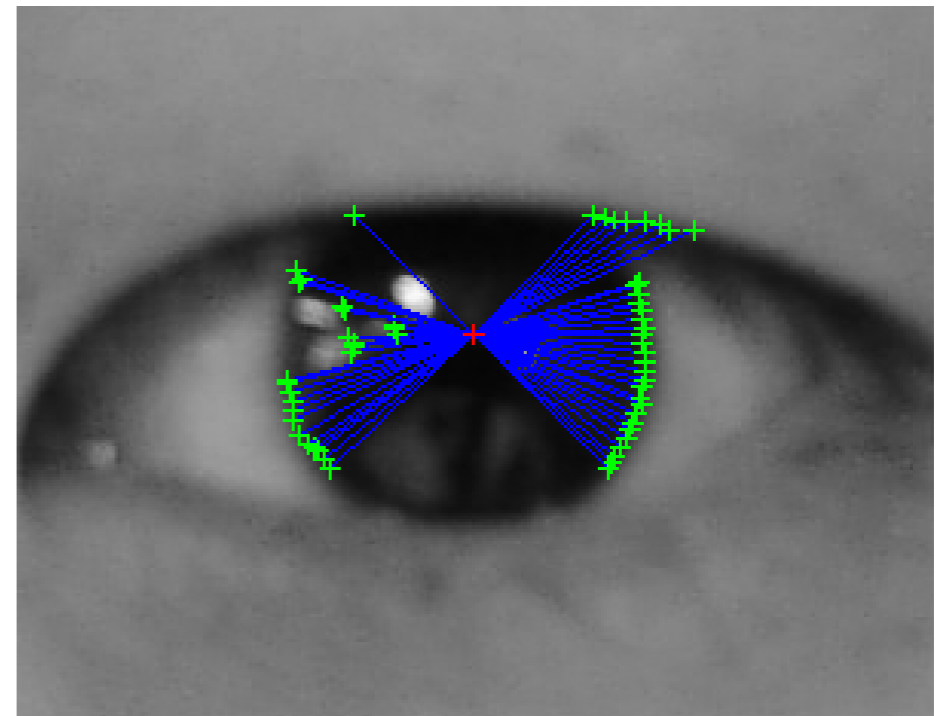


Architecture overview



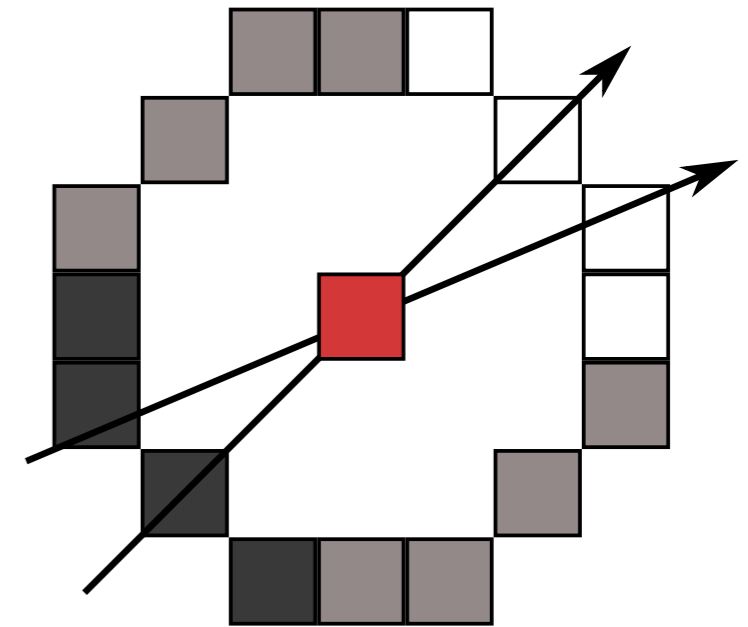
Starburst feature detection

- Find the closest features for each angle by exploring in a radial fashion from start point
- Random memory access
- It is not suitable for stream-oriented architecture
- Single-step starburst feature detection

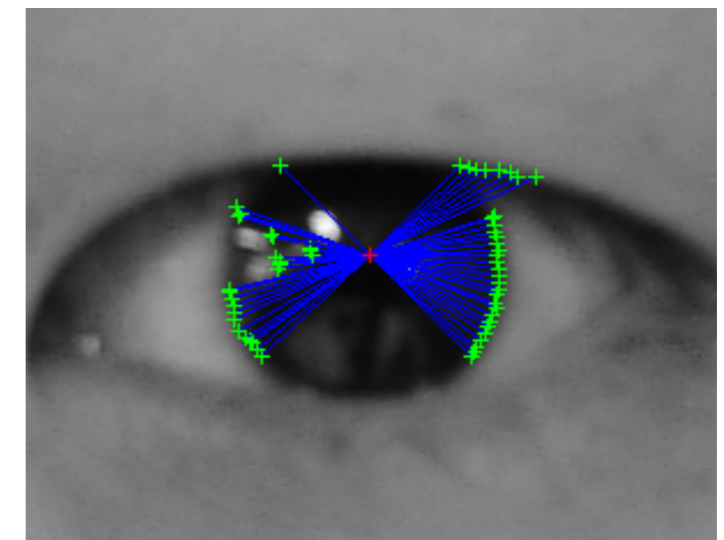


Stream-oriented feature detection

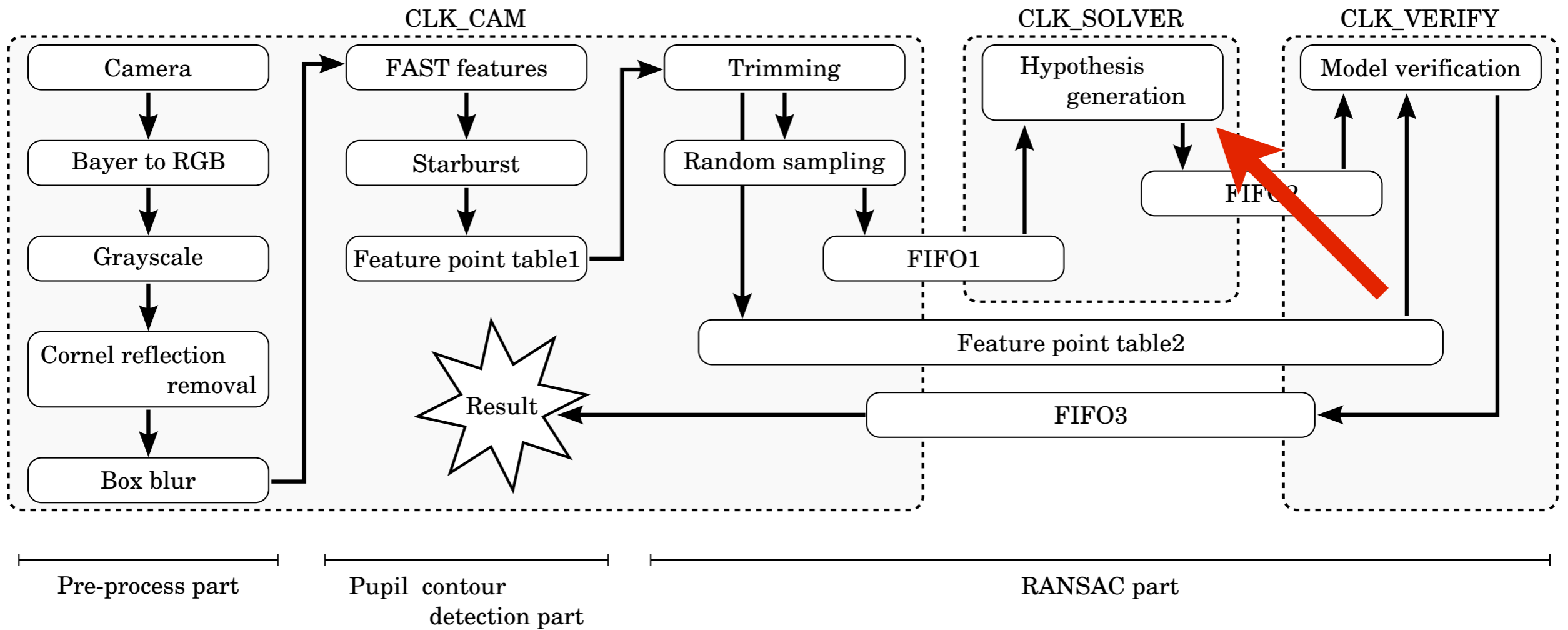
- Three-part processing for each pixel
 1. Calculate intensity derivatives
 2. Calculate distance and angle from the start point
 3. Find up to 128 feature points using gradient, angle and distance
- 16bit intensity derivative information for each pixel



Compute gradient using segment test



Architecture overview



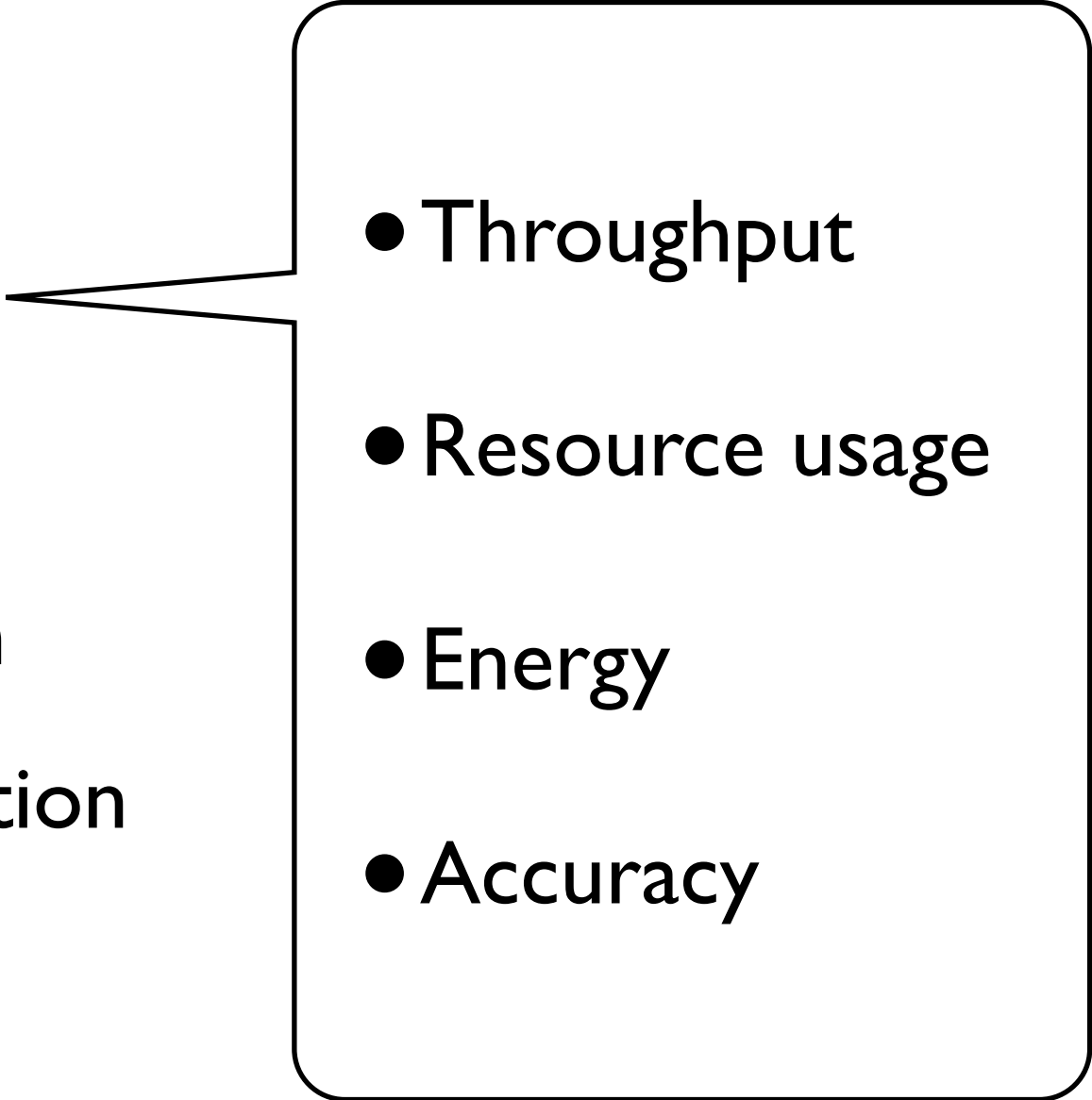
Ellipse estimation using feature points

1. Sample five points from the feature points set randomly
2. Generate ellipse from the five points
3. Check the error between the ellipse and feature point and count a number of points which has lower error than threshold (inliers)
4. Select a ellipse which has maximum number as a final result

$$\begin{pmatrix} \sum x_i^2 y_i^2 & \sum x_i y_i^3 & \sum x_i^2 y_i & \sum x_i y_i^2 & \sum x_i y_i \\ \sum x_i y_i^3 & \sum y_i^4 & \sum x_i y_i^2 & \sum y_i^3 & \sum y_i^2 \\ \sum x_i^2 y_i & \sum x_i y_i^2 & \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i^2 & \sum y_i^3 & \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i y_i & \sum y_i^2 & \sum x_i & \sum y_i & \sum 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \end{pmatrix} = \begin{pmatrix} -\sum x_i^3 y_i \\ -\sum x_i^2 y_i^2 \\ -\sum x_i^3 \\ -\sum x_i^2 y_i \\ -\sum x_i^2 \end{pmatrix}$$

Solving simultaneous equation for ellipse estimation

- 3 types of solvers
 - Cramer's rule
 - Gauss-Jordan elimination
 - Doolittle LU decomposition

- 
- Throughput
 - Resource usage
 - Energy
 - Accuracy

Cramer's rule

- Huge computational cost in proportion to $(n!)$
- An implementation of 4x4 matrix inversion using Cramer's rule and SIMD instructions is reported
- How about $n=5$ by FPGA?

$$Ax = b$$

$$x_i = \frac{|A_i|}{|A|}$$

$$|A| = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n A_{i,\sigma_i}$$

Expansion of Cramer's rule

$$\begin{aligned} |A| = & a[1][1] * a[2][2] * a[3][3] * a[4][4] * a[5][5] \\ & - a[1][1] * a[2][2] * a[3][3] * a[4][5] * a[5][4] \\ & - a[1][1] * a[2][2] * a[3][4] * a[4][3] * a[5][5] \\ & + a[1][1] * a[2][2] * a[3][4] * a[4][5] * a[5][3] \\ & + a[1][1] * a[2][2] * a[3][5] * a[4][3] * a[5][4] \\ & \vdots \end{aligned}$$

Expansion of Cramer's rule

$$\begin{aligned} |A| = & a[1][1] * a[2][2] * a[3][3] * a[4][4] * a[5][5] \\ & - a[1][1] * a[2][2] * a[3][3] * a[4][5] * a[5][4] \\ & - a[1][1] * a[2][2] * a[3][4] * a[4][3] * a[5][5] \\ & + a[1][1] * a[2][2] * a[3][4] * a[4][5] * a[5][3] \\ & + a[1][1] * a[2][2] * a[3][5] * a[4][3] * a[5][4] \\ & \vdots \end{aligned}$$

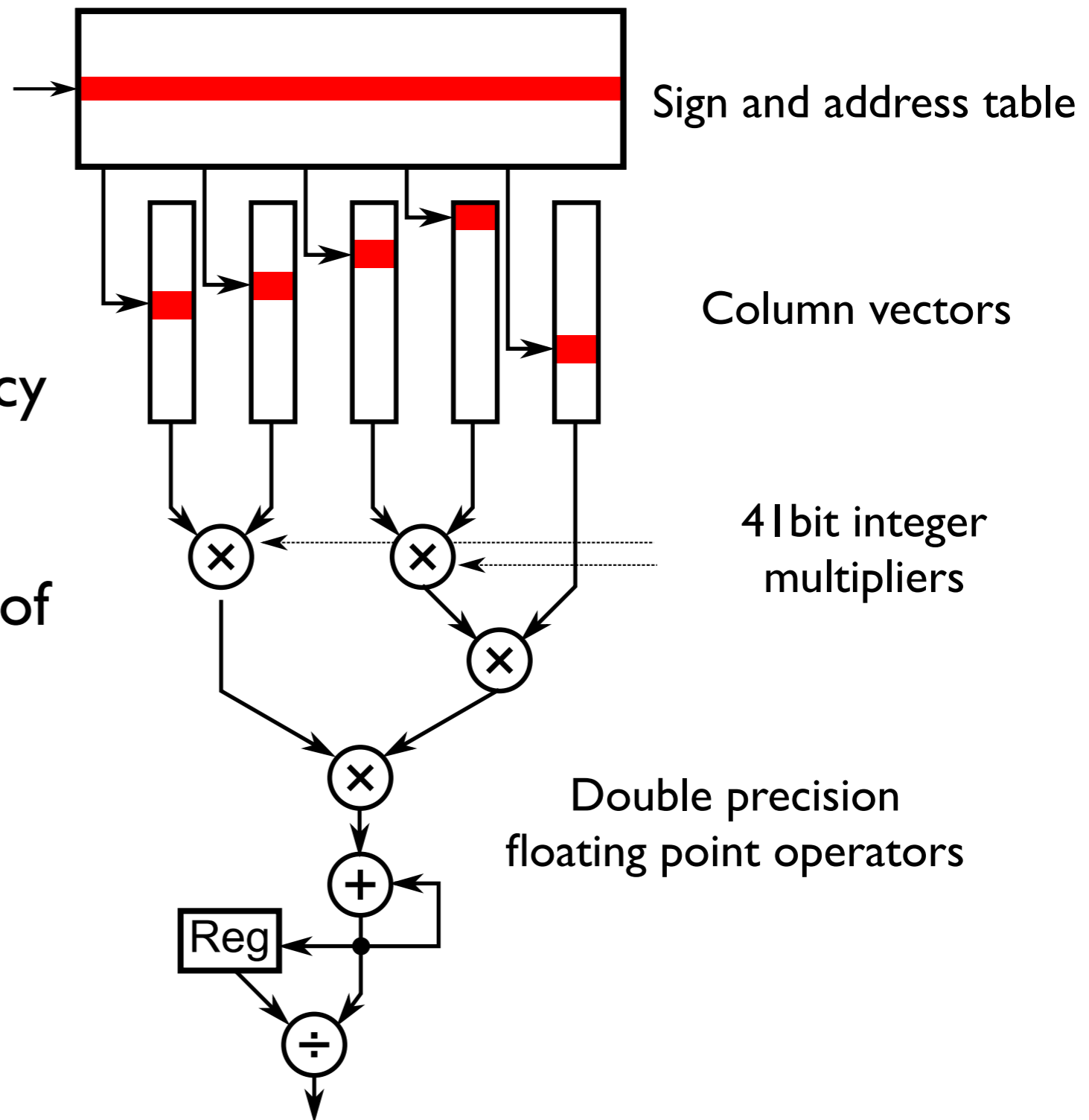
Store sign and row addresses on a ROM

Sign and row addresses table = 2Kb

Address(7bit)	Data(16bit)
0	+, 1, 2, 3, 4, 5
1	-, 1, 2, 3, 5, 4
2	-, 1, 2, 4, 3, 5
...	
119	+, 5, 4, 3, 2, 1

Overview of Solver_CRAMER

- Highly parallelism
- Interleave calculations of 6 determinants to hide latency of accumulator
- Mixed operations consists of integer and floating-point operators
- Double-precision floating point divider



Gauss-Jordan elimination

- Don't need backward substitution
- No pivoting
- Abandon a computational result when something is wrong
- Throughput, throughput and throughput

$$Ax = b$$



$$Ex = b'$$

Recurrence formula for pipelining

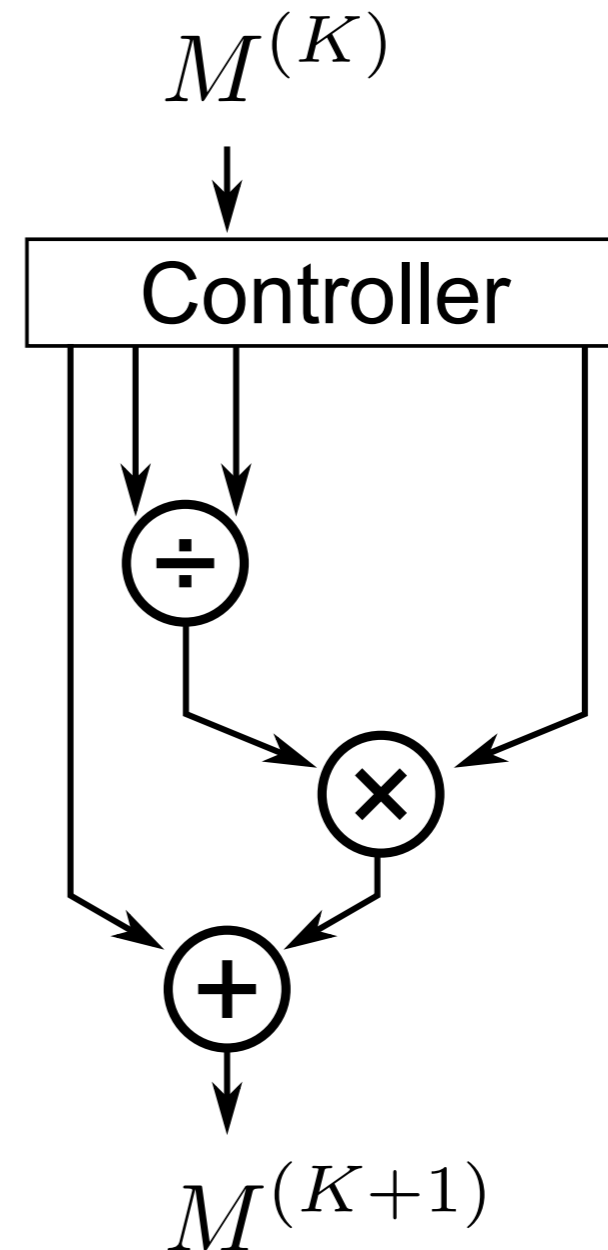
Calculate $M^{(5)}$ using :

$$M^{(0)} = [Ab]$$

$$m_{i,j}^{(K)} = \begin{cases} m_{i,j}^{(K-1)} & \text{if } j < K, \\ \frac{m_{i,j}^{(K-1)}}{m_{K,K}^{(K-1)}} & \text{else if } i = K, \\ m_{i,j}^{(K-1)} - \frac{m_{i,K}^{(K-1)}}{m_{K,K}^{(K-1)}} m_{K,j}^{(K-1)} & \text{otherwise.} \end{cases}$$

Overview of Solver_GAUSS

- Cascaded 5 sub-modules
- Each sub-module independent from others
- Single precision floating point operators



Doolittle LU decomposition

1. Decompose for two matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 & 0 \\ l_{41} & l_{42} & l_{43} & 1 & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} \\ 0 & 0 & u_{33} & u_{34} & u_{35} \\ 0 & 0 & 0 & u_{44} & u_{45} \\ 0 & 0 & 0 & 0 & u_{55} \end{bmatrix}$$

2. Calculate a result using forward and backward substitution

$$Ly = b$$

$$Ux = y$$

Recurrence formulas for pipelining

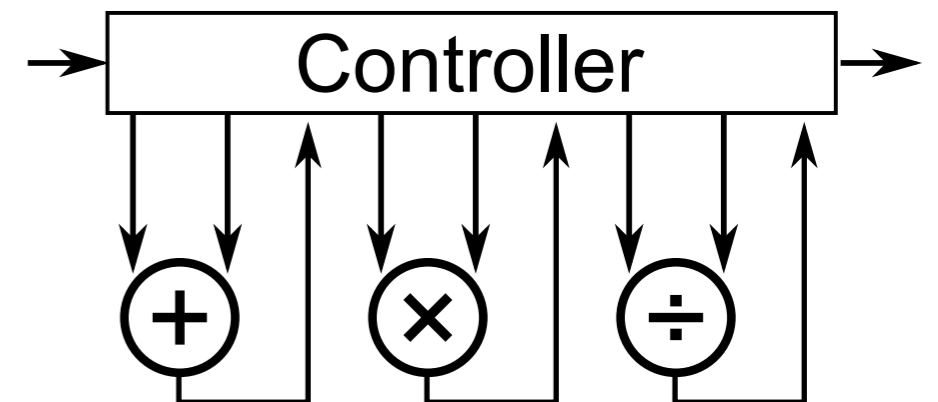
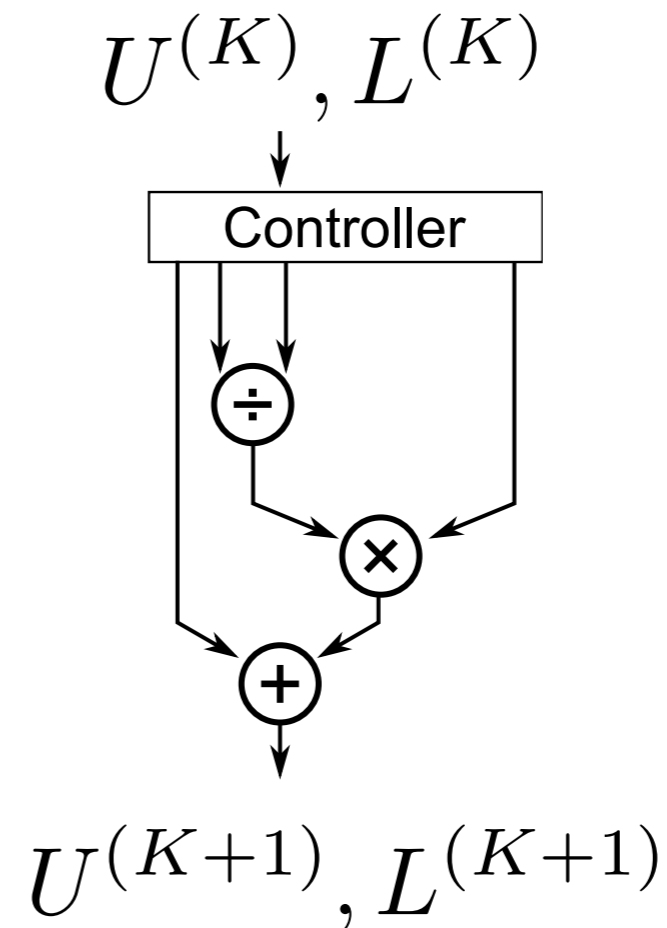
$$U^{(0)} = A$$

$$u_{i,j}^{(K)} = \begin{cases} u_{i,j}^{(K-1)} & \text{if } i \leq K, \\ u_{i,j}^{(K-1)} - \frac{u_{i,K}^{(K-1)}}{u_{K,K}^{(K-1)}} u_{K,j}^{(K-1)} & \text{otherwise.} \end{cases}$$

$$l_{i,j}^{(K)} = \begin{cases} l_{i,j}^{(K-1)} & \text{if } j < K, \\ \frac{u_{i,j}^{(K-1)}}{u_{K,j}^{(K-1)}} & \text{else if } i \geq K, \\ 0 & \text{otherwise.} \end{cases}$$

Overview of Solver_LU

- Use similar sub-module for Gauss-Jordan elimination
- Forward and backward substitutions need many clock cycles
- Single precision floating point operators



Environment

- ML501 evaluation board
- Xilinx Virtex-5 5XC5VLX50
- OmniVision OV9620 Cam
- ISE 13.4
- 1Ω Shunt resistor
- Pre-process, feature detection : 25MHz
- Ellipse estimation : 100MHz



Resource usages for each implementation

	CRAMER	GAUSS	LU	Available
FF	20,667	22,267	24,062	28,800
LUT	18,709	19,130	19,725	28,800
BRAM	39	39	34	48
DSP48	47	44	39	48

Resource usage for divider

	CRAMER		GAUSS		LU	
	FF	LUT	FF	LUT	FF	LUT
Solver	11,735	9,816	13,335	10,229	15,130	10,816
Divider	5,982	3,207	6,760	3,840	6,760	3,840
Ratio	51%	33%	51%	38%	47%	36%

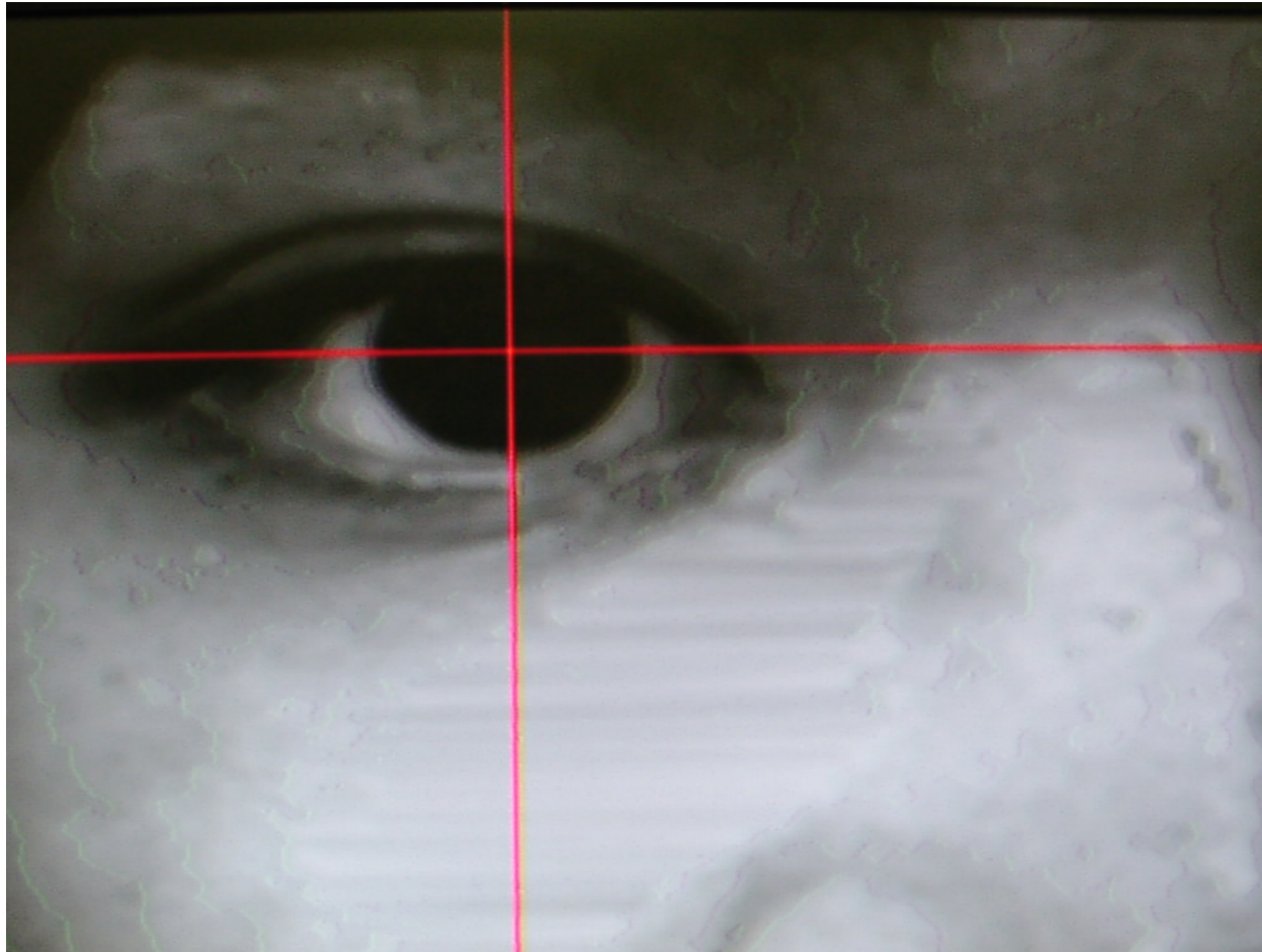
Dividers occupy large area in all the solvers

Power consumption and Performance

	CRAMER	GAUSS	LU
Total power [W]	3.34	3.29	3.16
Solver Power [W]	0.33	0.28	0.15
Latency [us]	8.69	3.05	4.99
Throughput	1.1E+06	1.0E+07	3.8E+06

Big difference in throughput

Example



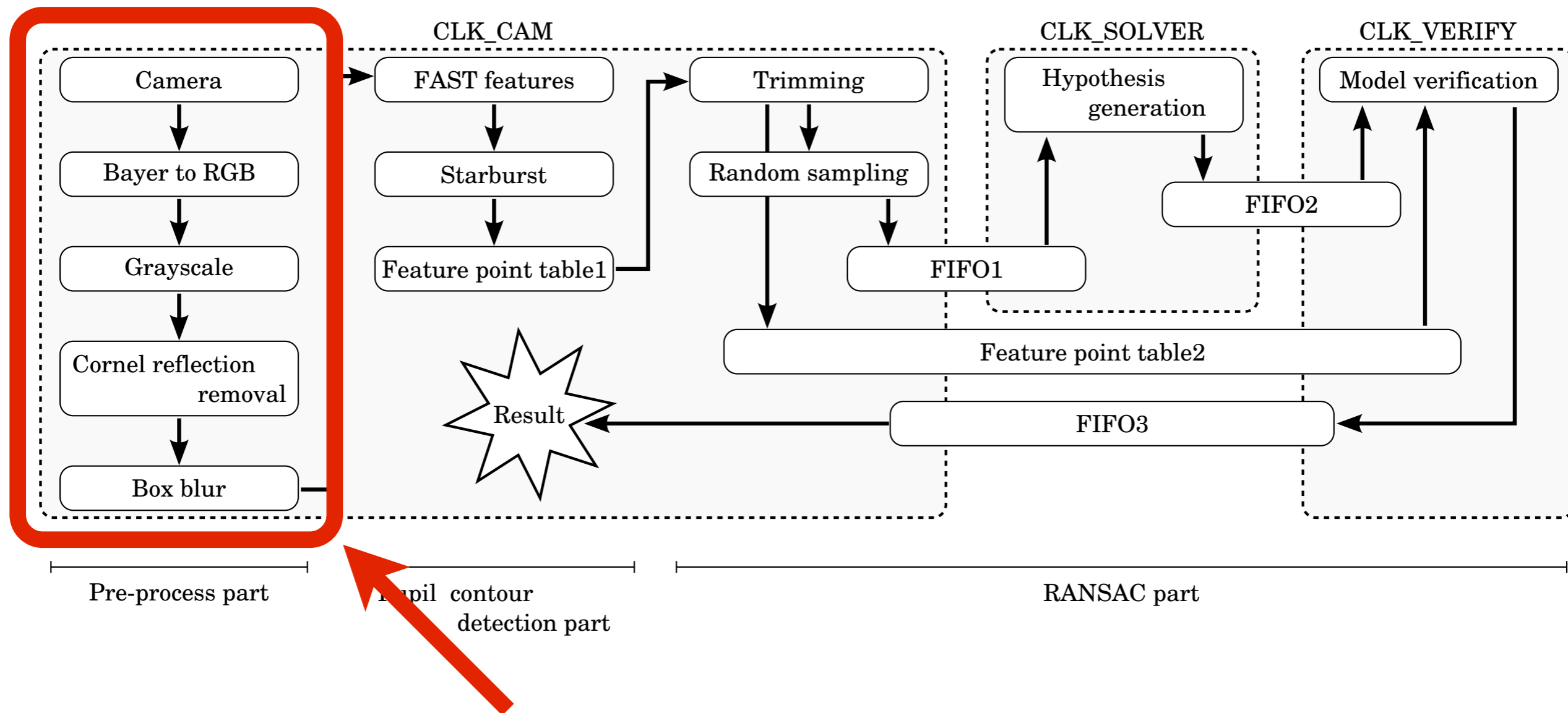
Conclusion

- Hardware implementation for ellipse estimation without using any external memory
- Stream-oriented pre-processes and feature detection
- Compare 3 types of solver for ellipse estimation
 - Cramer's rule: Lowest FF and LUT usage
 - Gauss-Jordan elimination : Higher throughput
 - LU decomposition : Lower power consumption
- Dividers occupy large area in all the designs
- Accuracy evaluation (Future)
- More compact design (Future)

Related research

- Martelli et al, FPGA-Based Robust Ellipse Estimation for Circular Road Sign Detection
 - FPGA implementation for ellipse detection algorithm
 - Used simplified ellipse equation

Architecture overview



Pre-processes

1. Get image from camera (Bayer pattern)
2. Convert to gray scale
3. Remove reflection
4. Apply box-blur filter

