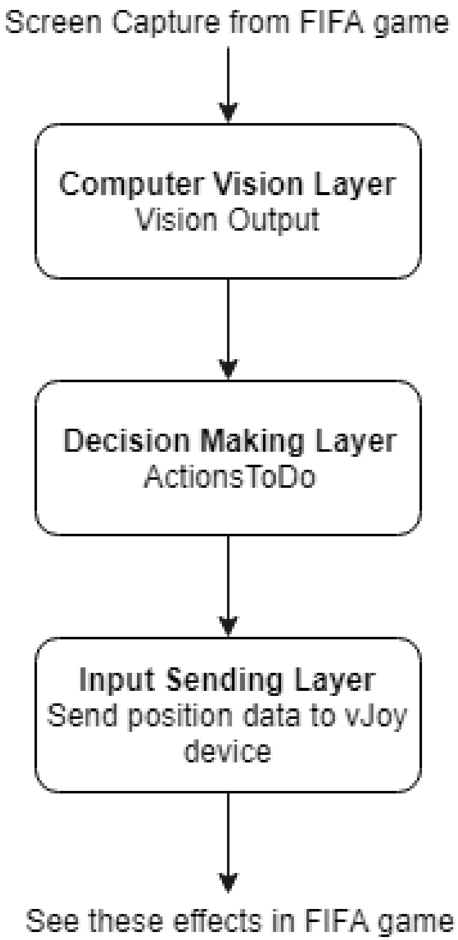
AI pentru fotbal utilizand computer vision

Obiectivul: AI pentru FIFA

- Detectarea comportamentului echipei adverse si in functie de acesta, sa se efectueze actiuni precum: pase, sut, atac ori dribbling.
- Aplicarea acestor actiuni in jocul FIFA – analizand pozitia jucatorilor din ambele echipe, pozitia mingii, detectarea marcajelor de pe teren, dar si a eliminarii unor elemente precum: multimea de oameni (crowd-ul).
- Nu este nevoie de codul jocului pentru antrenare.
- Incercari similare: Dota2, dar in cazul acestuia este ceva mai simplu fiind un joc 2D, destul de usor de inteles deciziile.

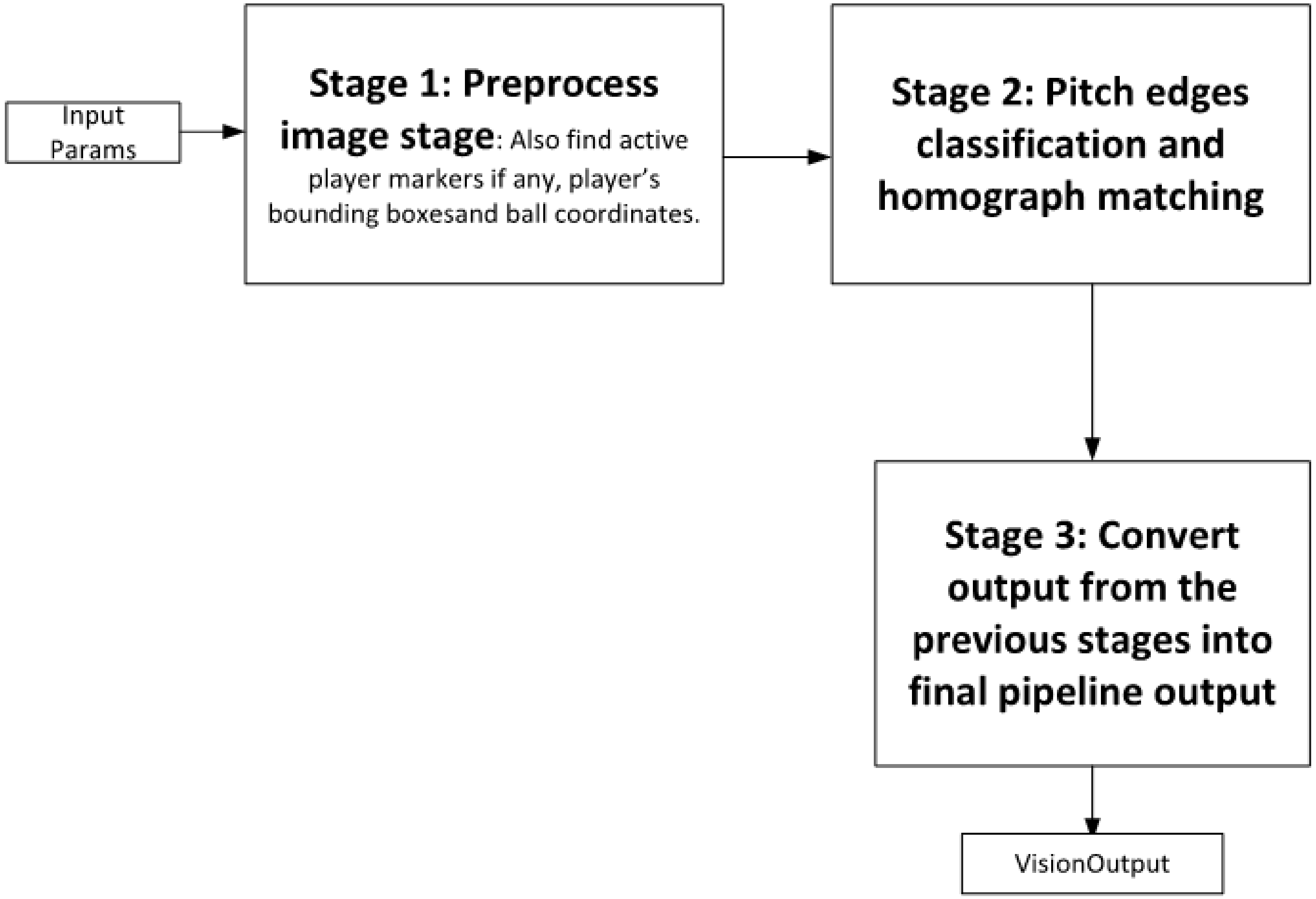
Aplicatie

- Aplicatia a fost scrisa in C/C++
- Contine 3 layere
- S-a folosit OpenCV – o biblioteca de functii open source (C/C++) folosita pentru aplicatii de computer vision in timp real.
- De asemenea pentru procesarea actiunilor transmise de layer-ul de Computer Vision se foloseste un joystick virtual: vJoy – proiect integrat in aplicatie.

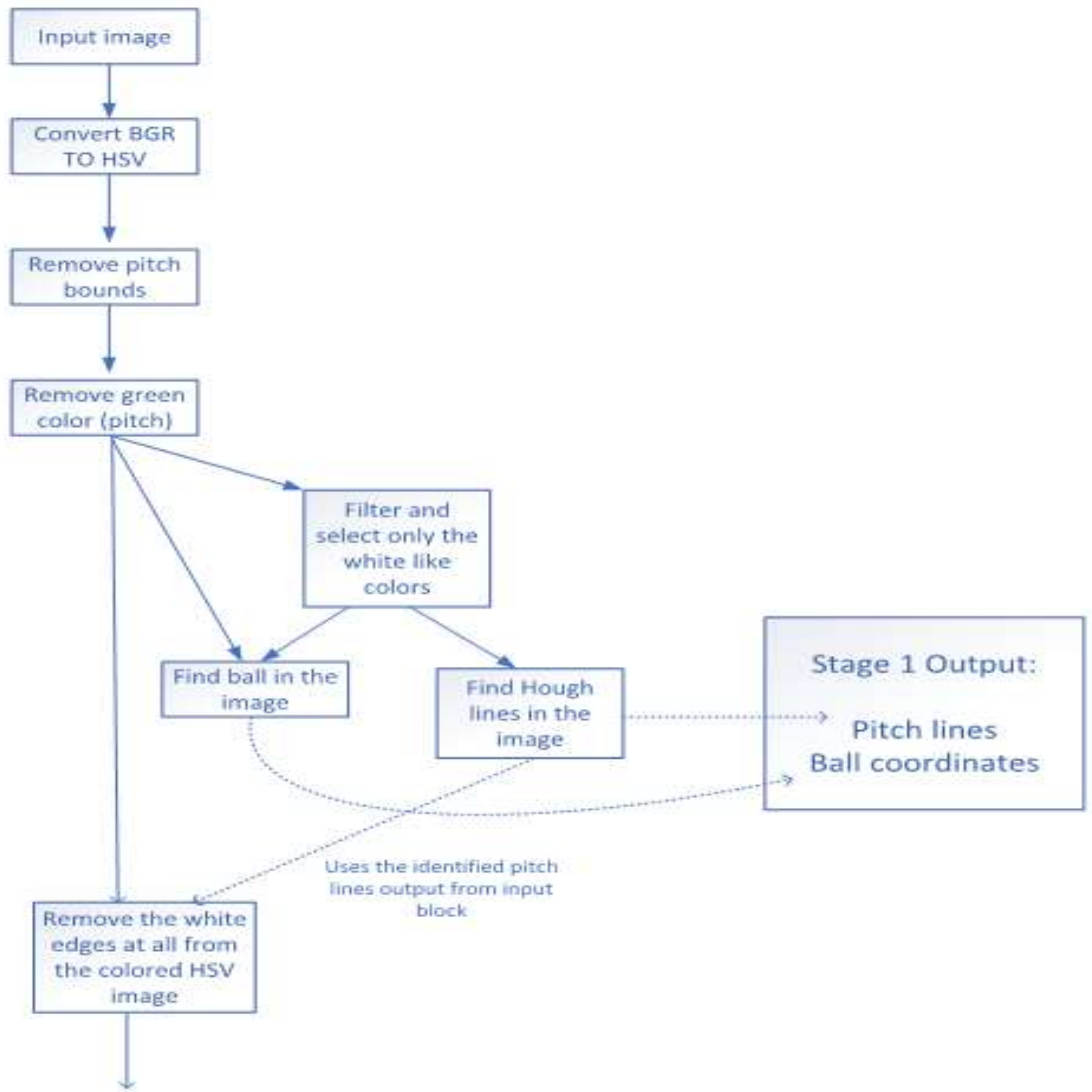


Computer Vision Layer

- Este un proces pipeline impartit in 3 stagii:



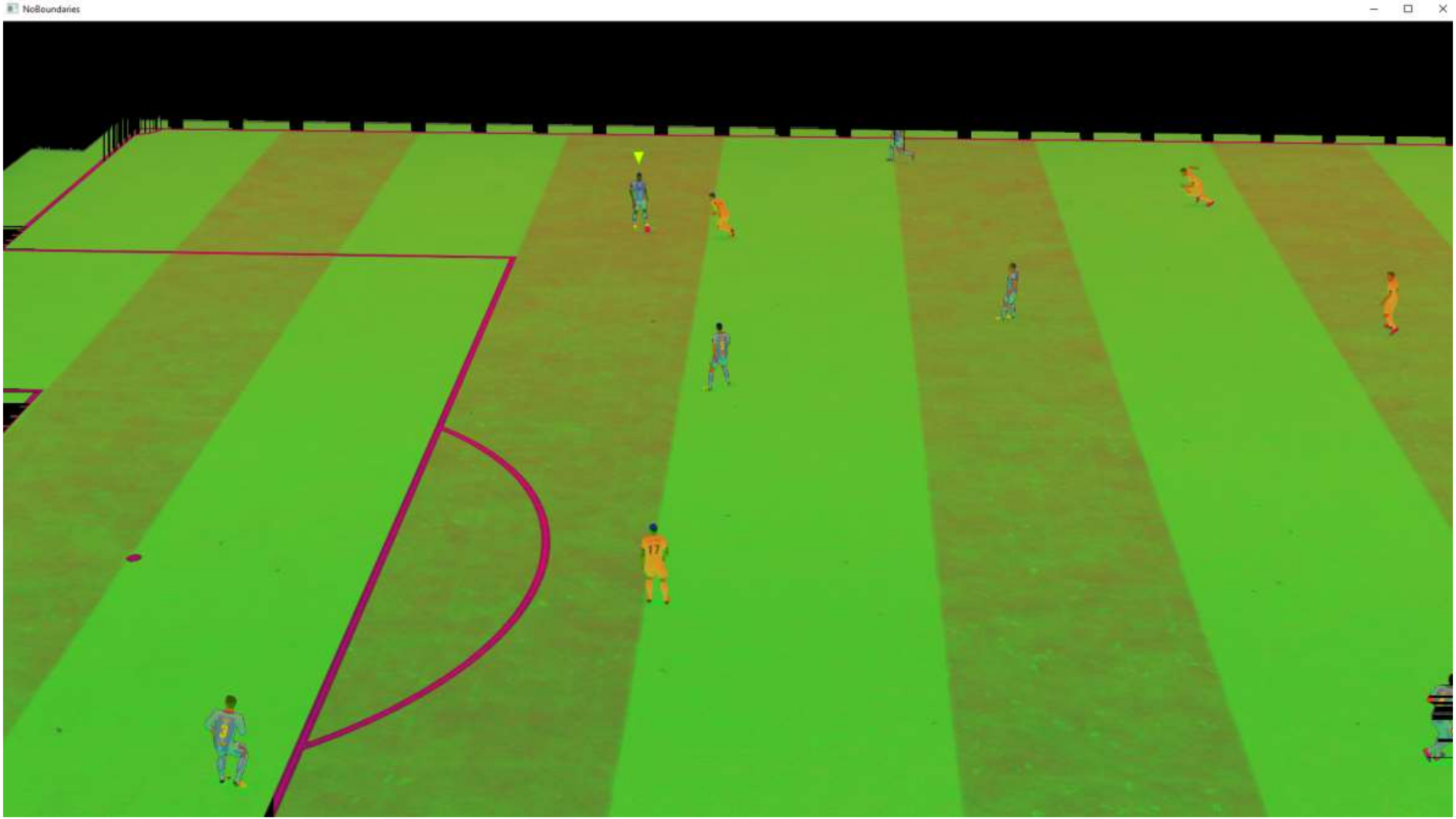
Computer Vision Layer – Stagiul 1



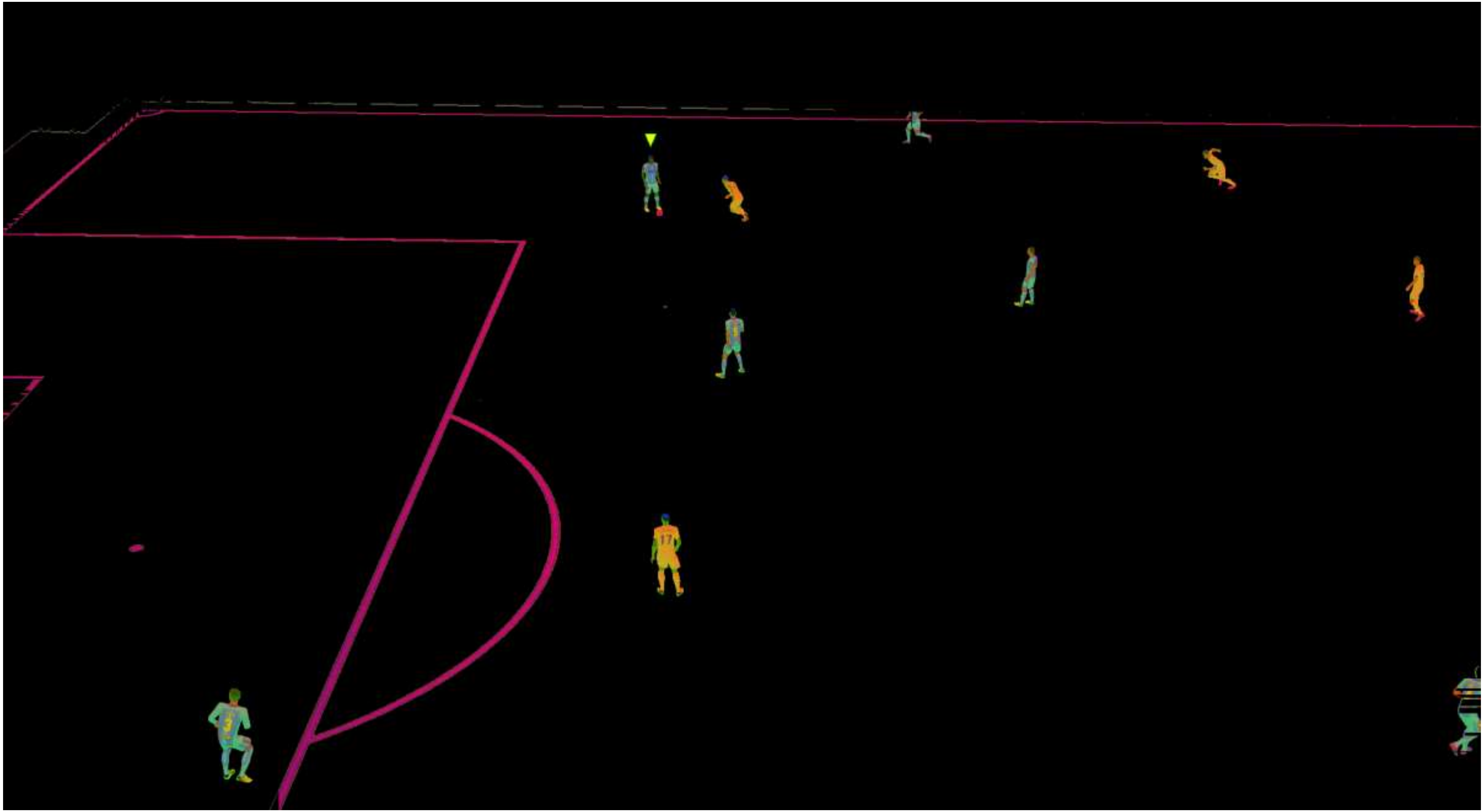
Input Image

HSV Space

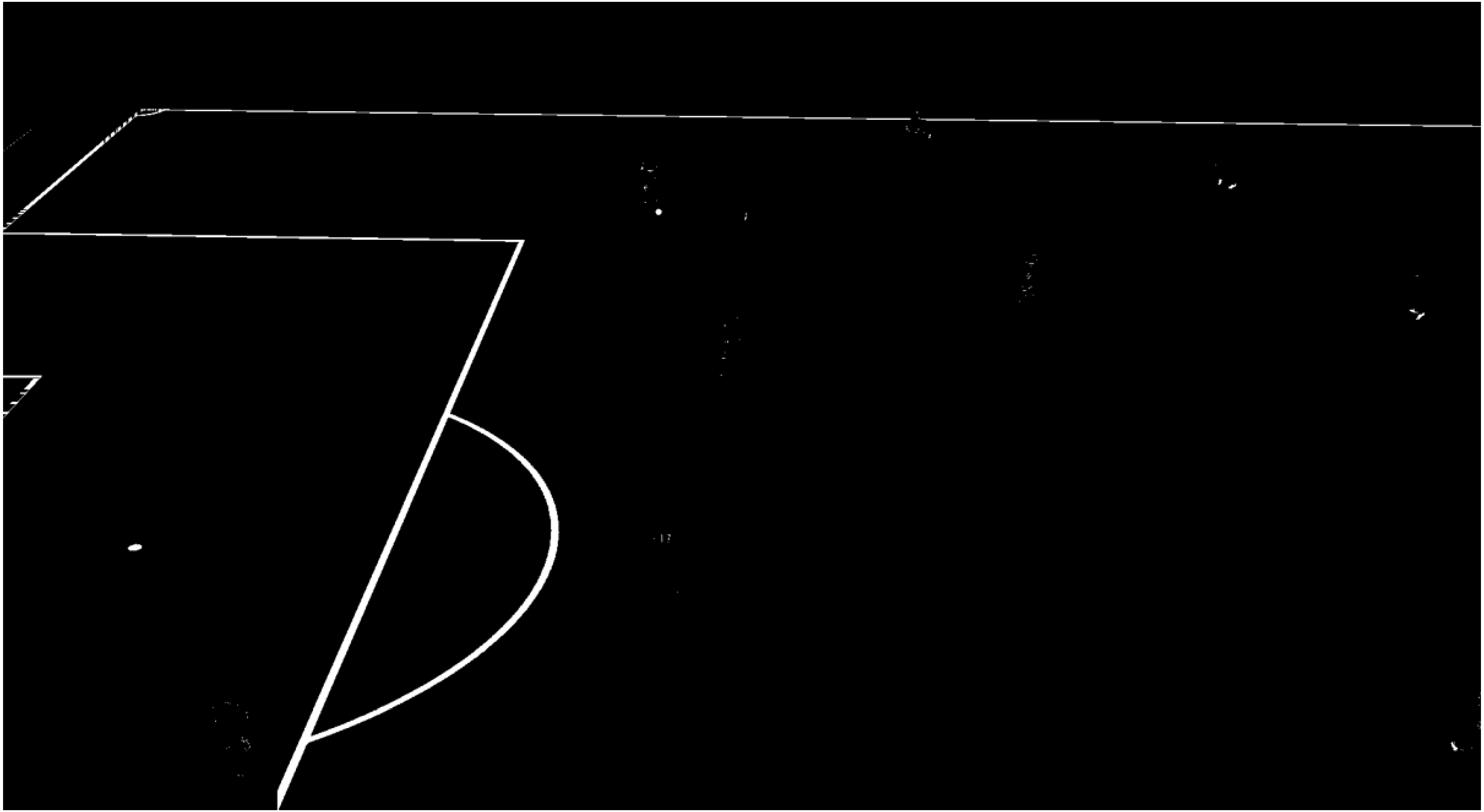
Crowd elimination



No green

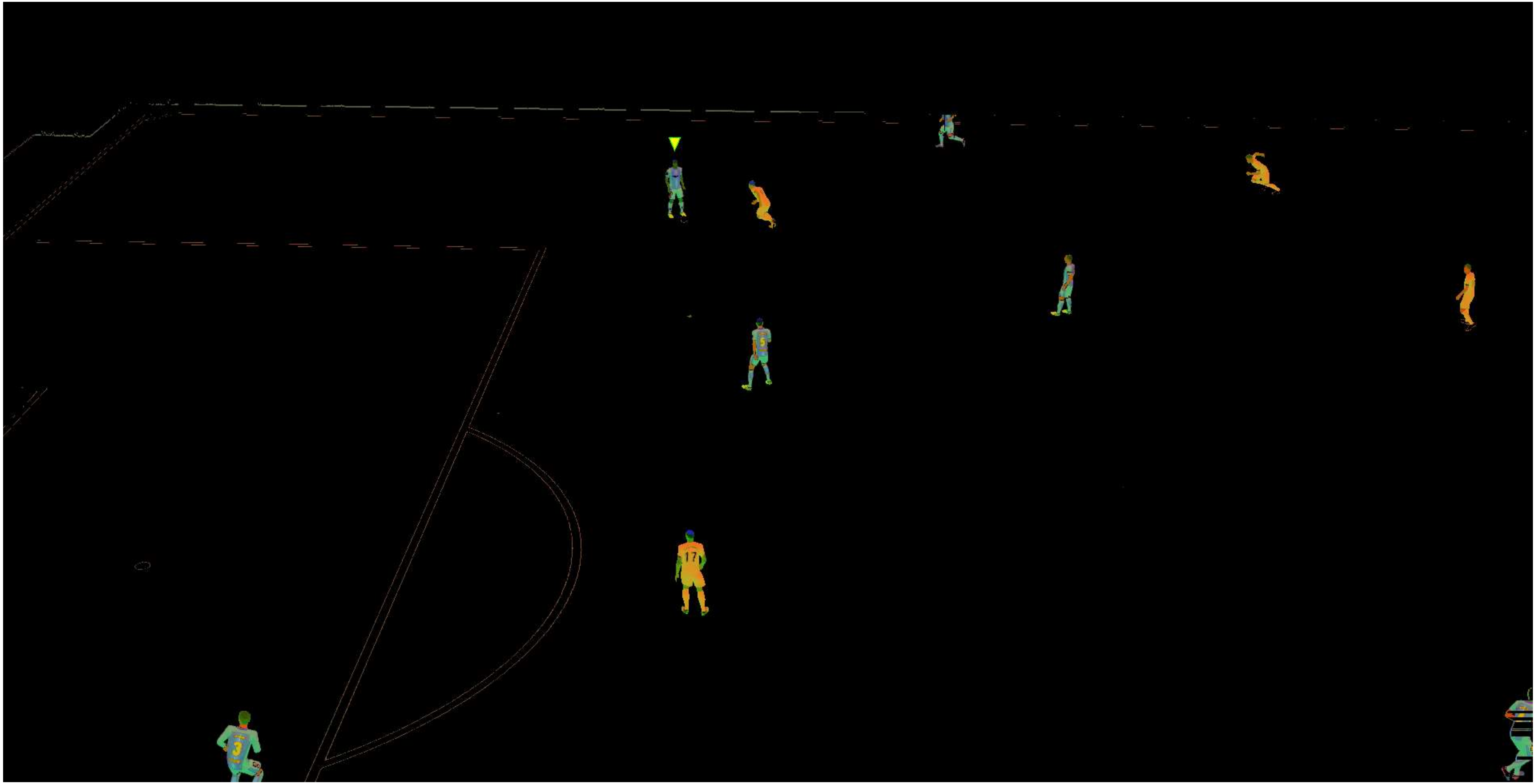


HSV Mask – only white

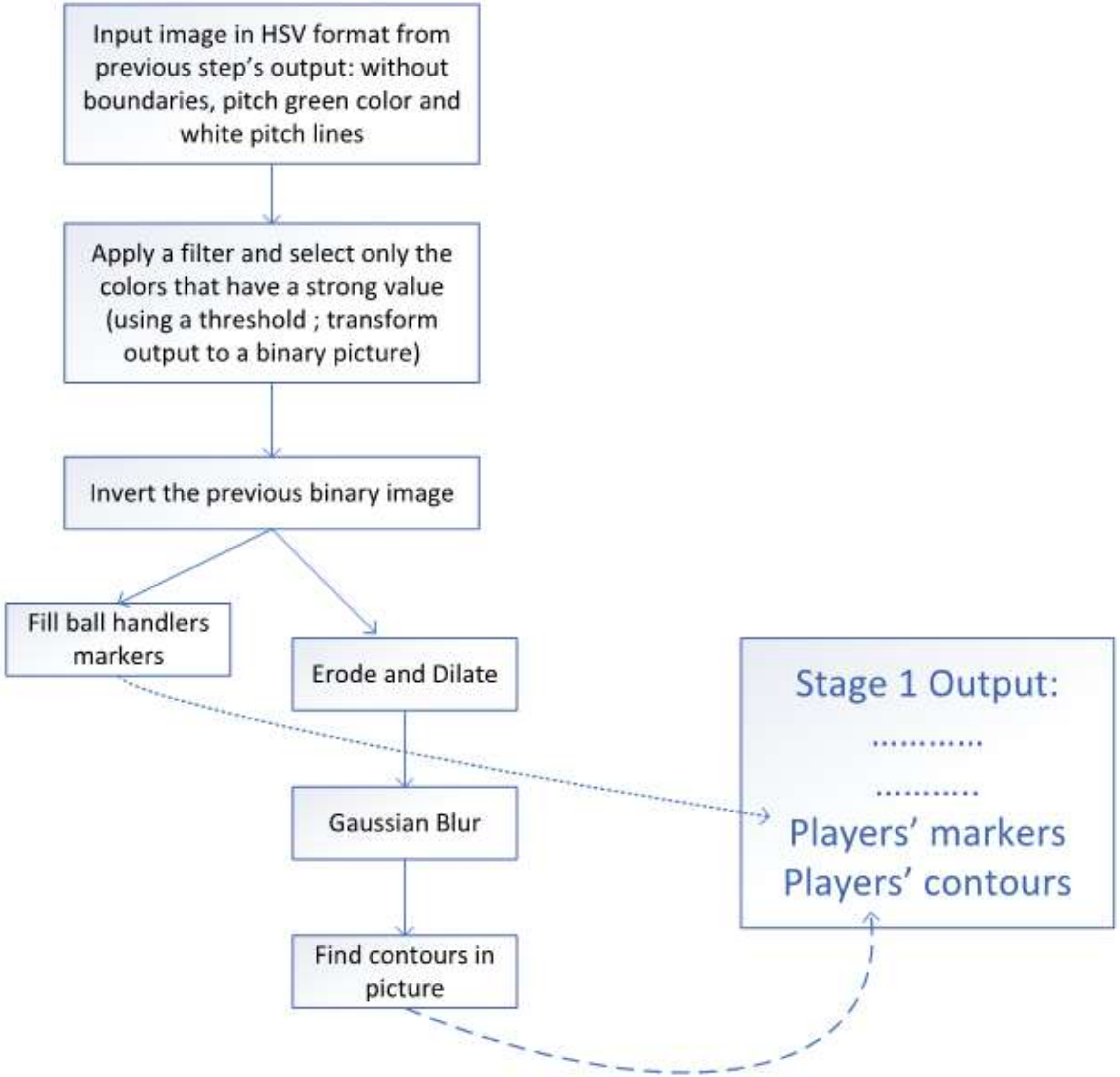


Hough Lines

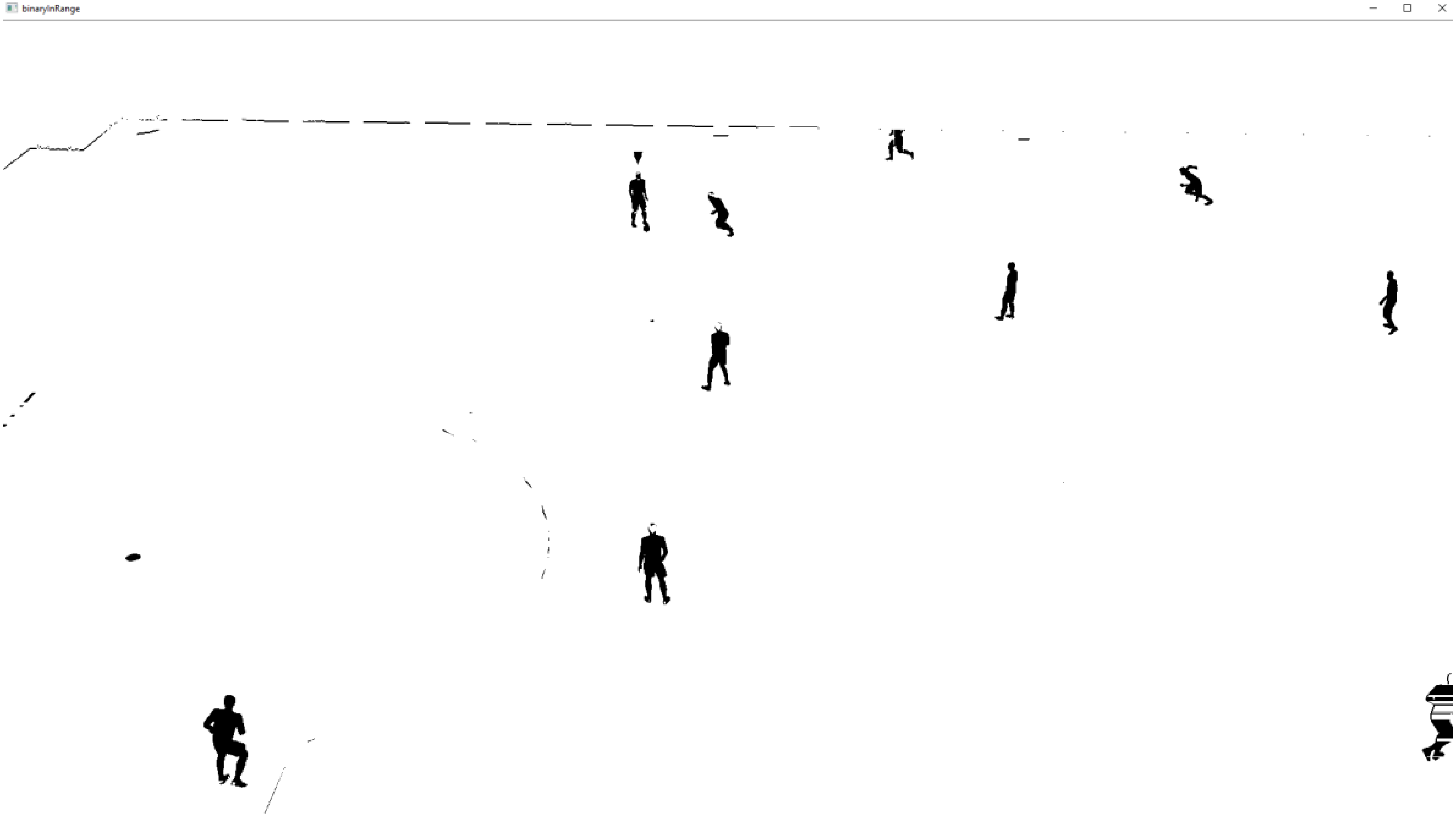
No lines



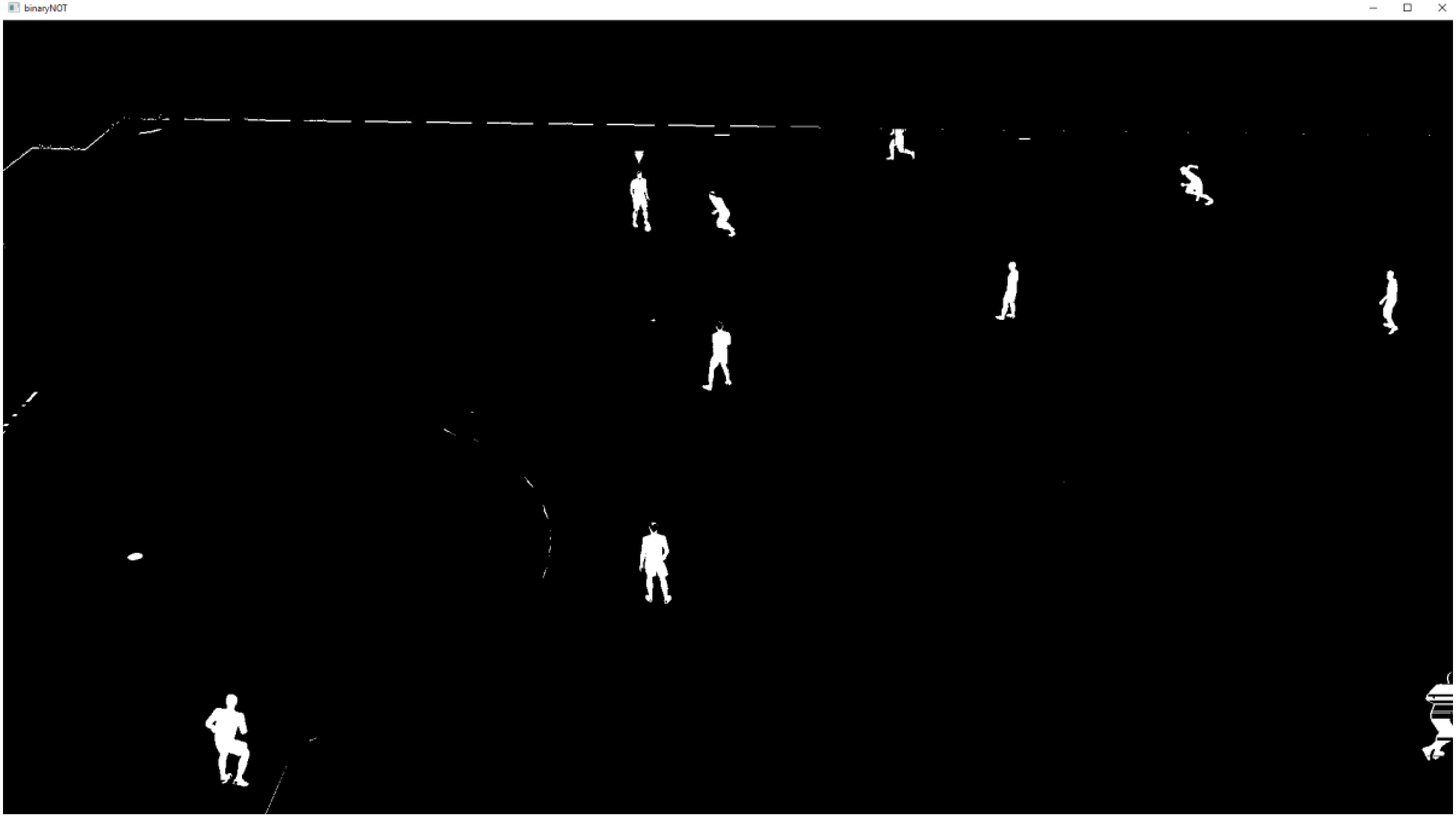
Computer Vision Layer – Stagiul 1



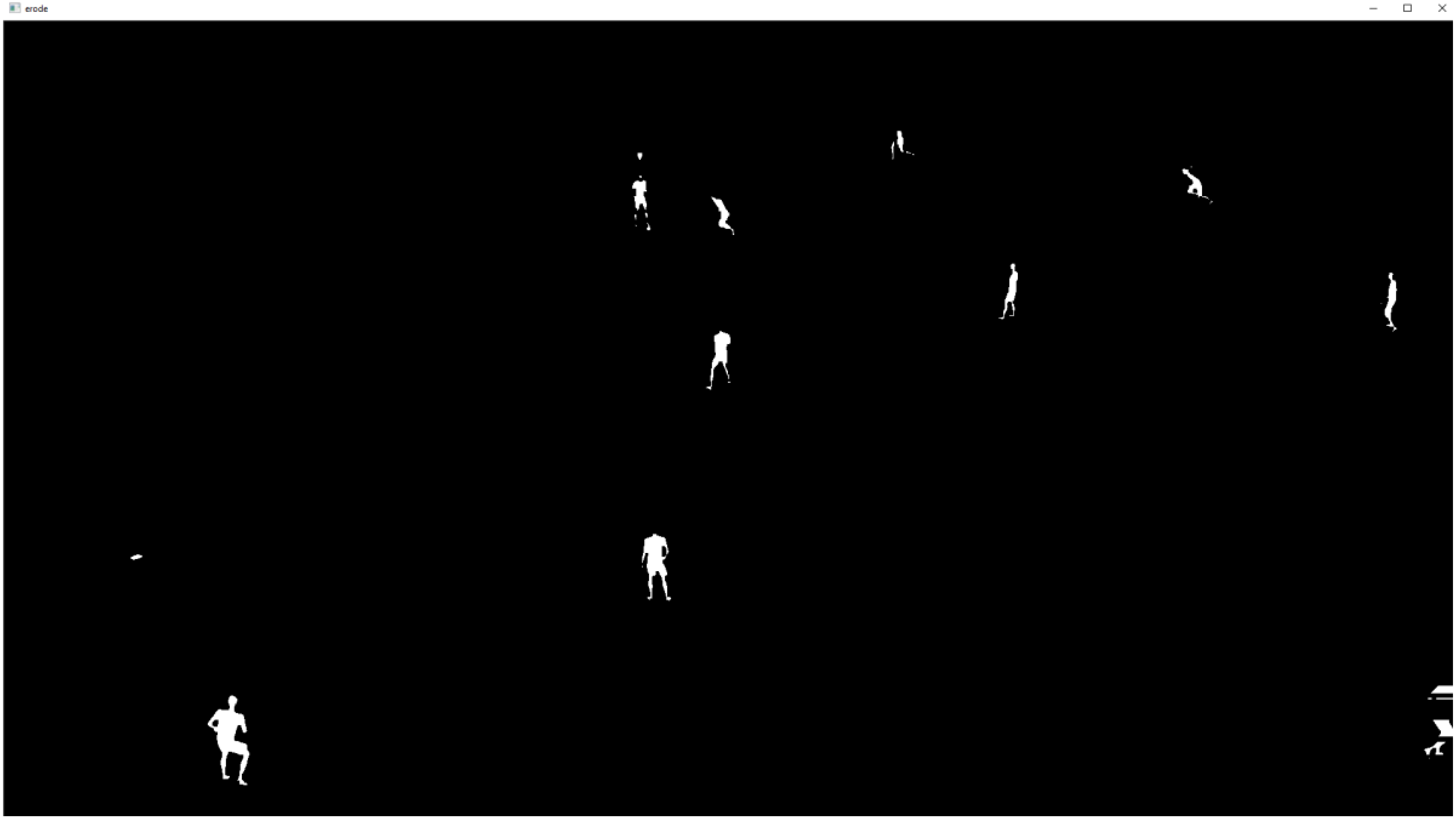
Binary Image



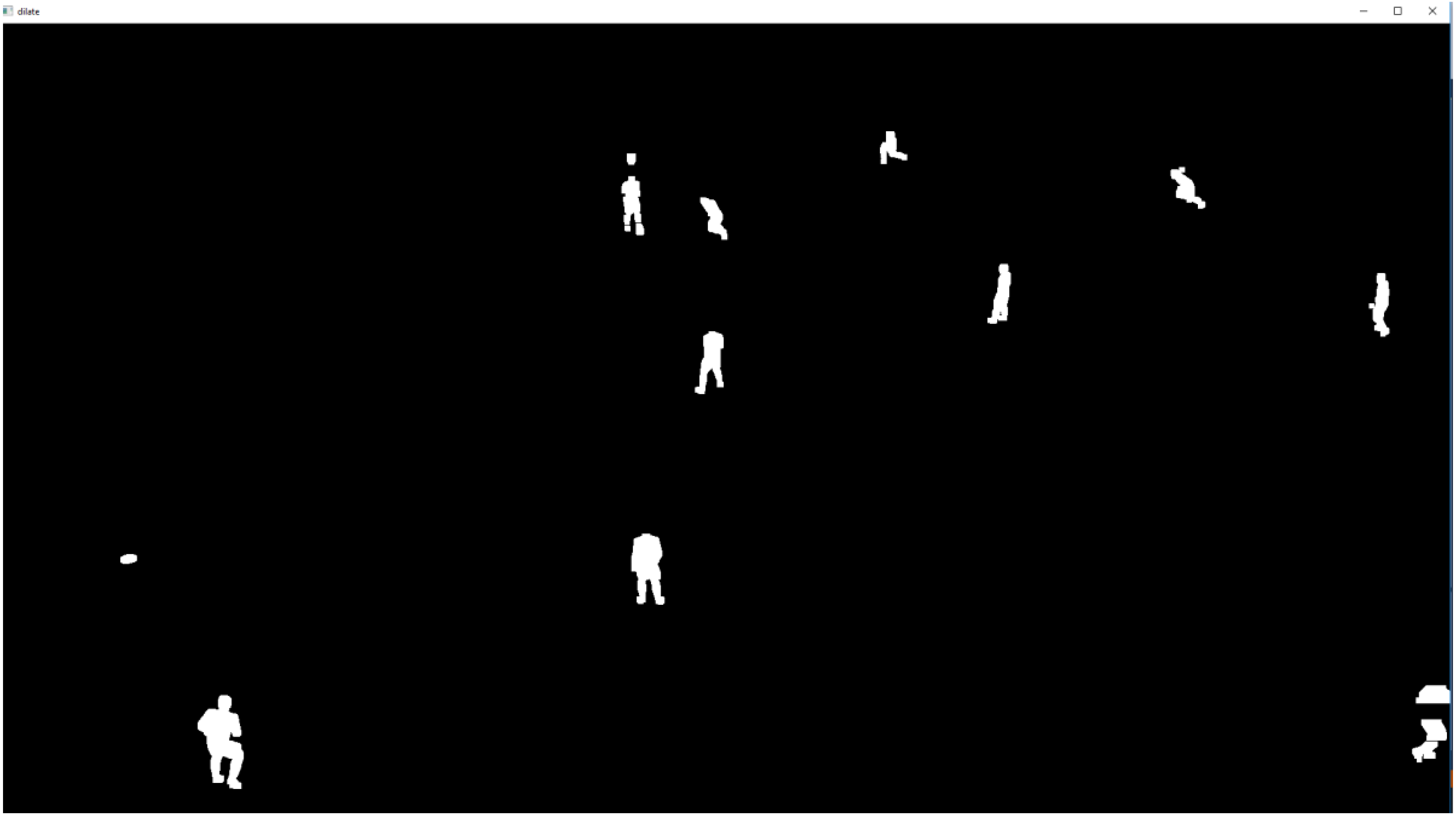
Not Binary



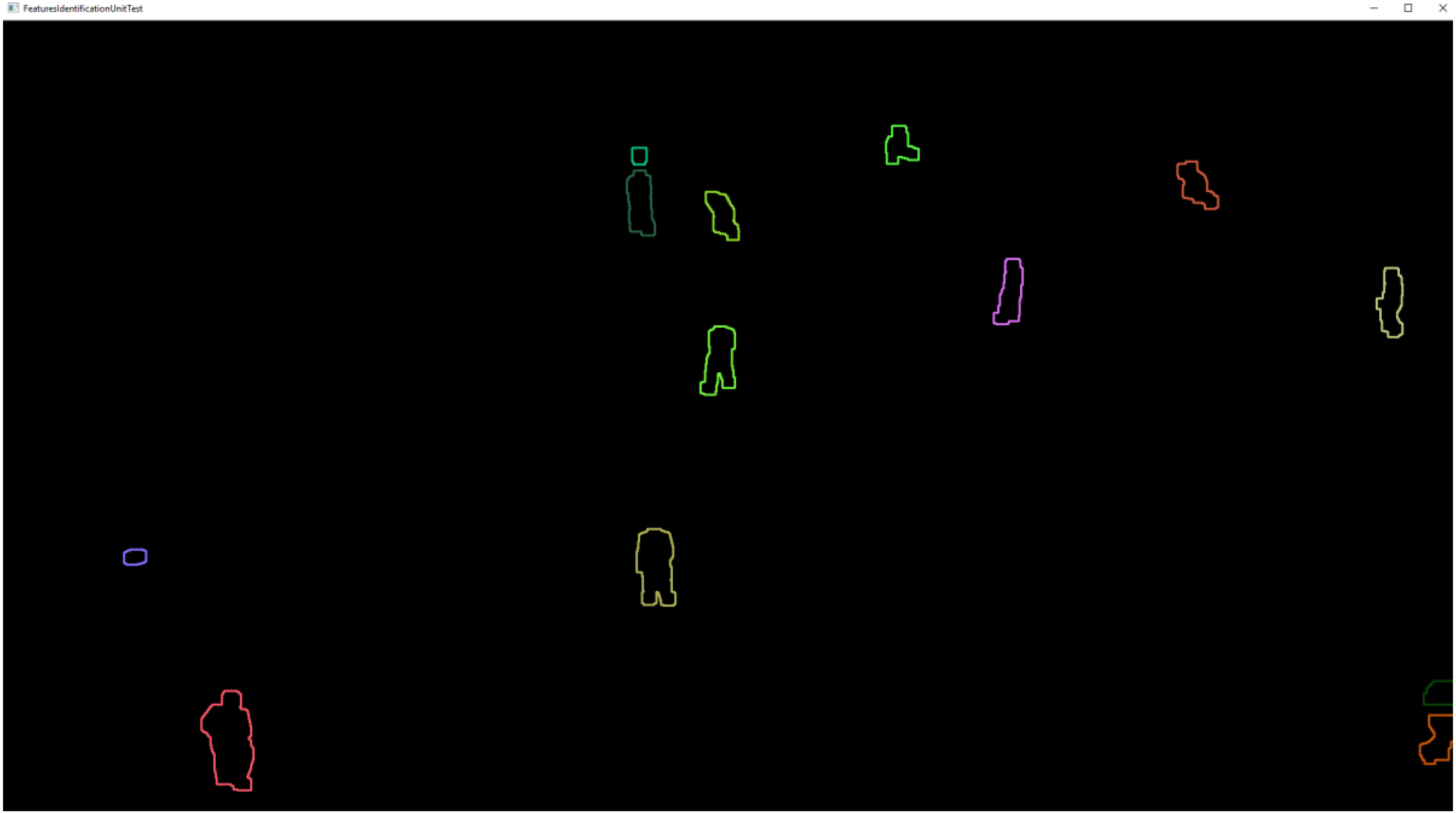
Erosion



Dilation



Contours



Computer Vision Layer – Stagiul 2

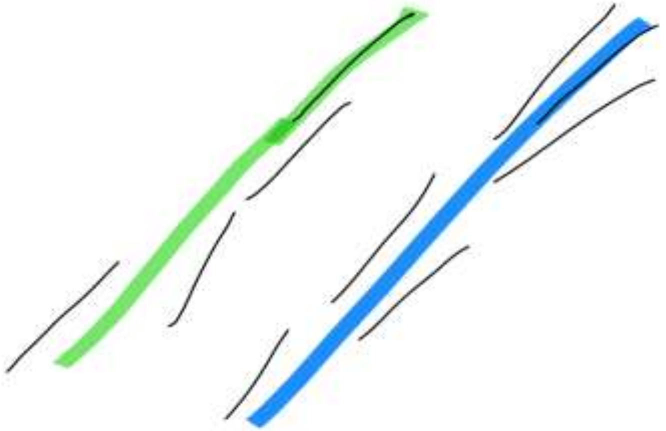
2.1. Se clasifica segmentele in 2 seturi : orizontale si vertical =>Output: seturile *HorizontalSegments*, *VerticalSegments*;

```
For each segment S in InputSegments
    If slope(S) > T
        VerticalSegments.add(S)
    Else
        HorizonalSegments.add(S)
// Computes the histograms of slopes and find the longest bin cluster
HistSlopesHorizontal = Histogram of slopes in HorizonalSegments
HistSlopesVertical = Histogram of slopes in VerticalSegments
DominantBinHorizonal = HistSlopesHorizonal.argmax(total dimension of segments in
bin);
DominantBinVertical = HistSlopesVertical.argmax(total dimension of segments in bin);

// Keep only the segments in the dominant cluster
HorizonalSegments = HistSlopesHorizonal[DominantBinHorizonal]
VerticalSegments = HistSlopesVertical[DominantBinVertical]
```

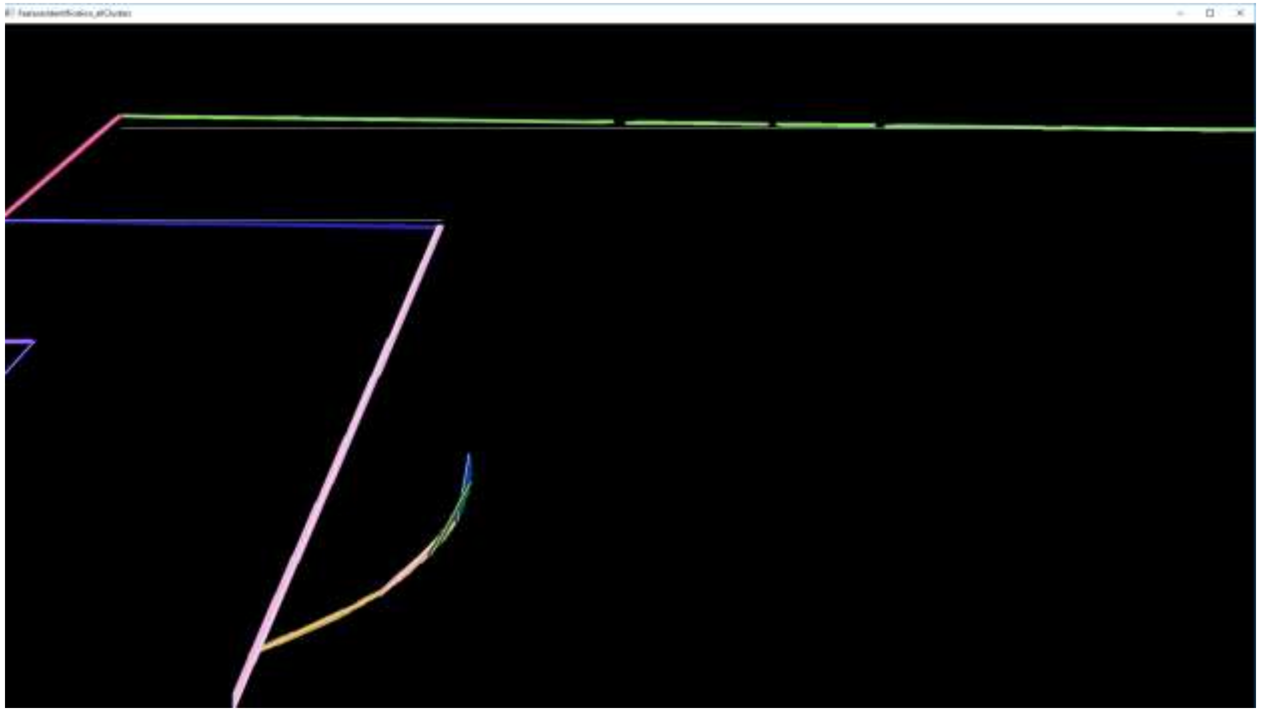
Computer Vision Layer – Stagiul 2

2.2. Se pun in clustere segmentele care fac parte din acelasi feature de pe teren (endline, sidelines, box, small box, middle line), verificand atat pentru *HorizontalSegments*, cat si *VerticalSegments*

Noise elimination

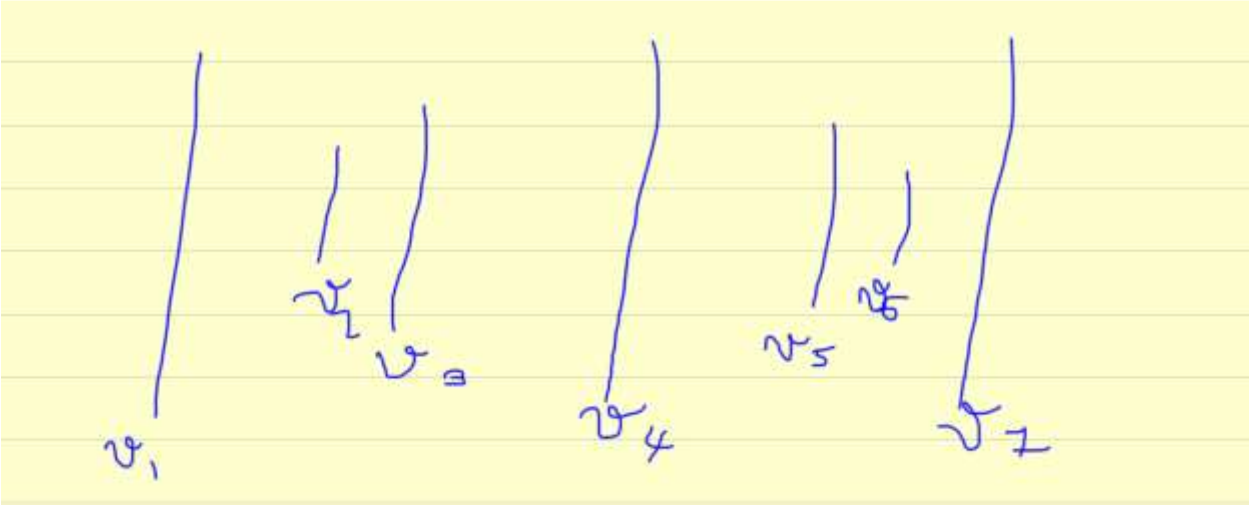


Clusterization



Computer Vision Layer – Stagiul 2

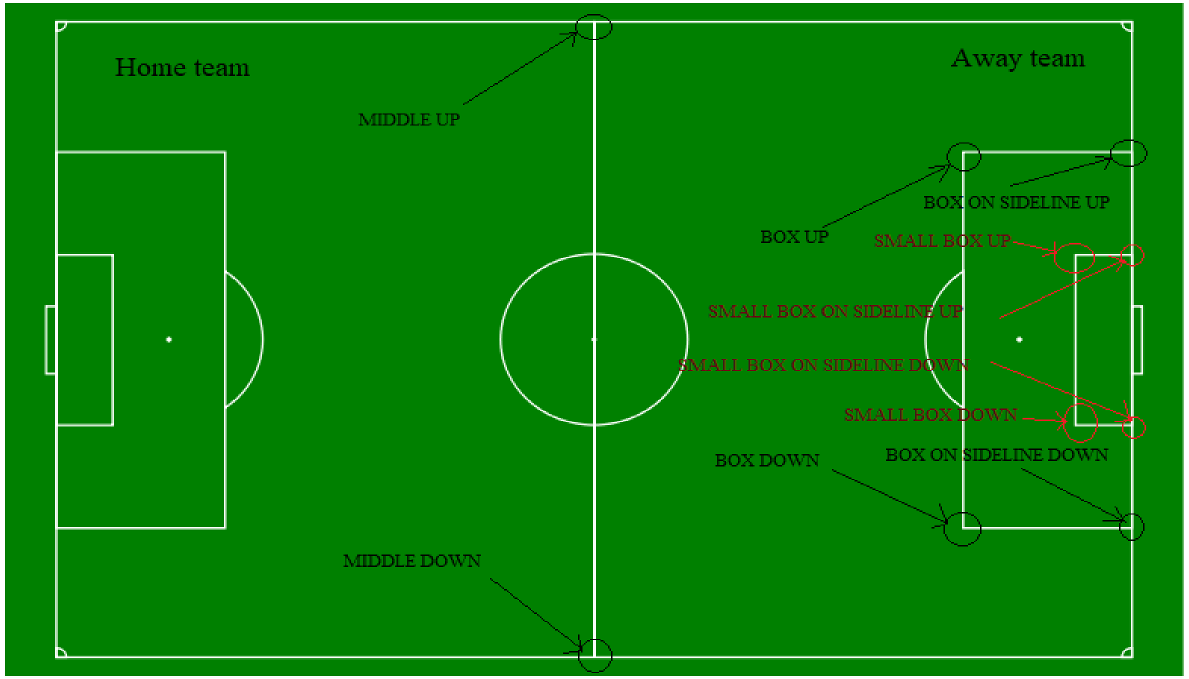
2.3. Se selecteaza cele mai vizibile clustere – pentru a gasi feature points

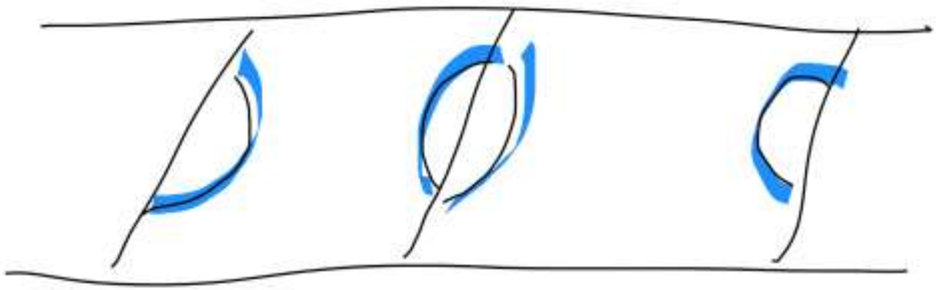


CMax = select the longest representative visible cluster in VerticalClusters
For each cluster C in Cluster different than CMax
    If C is parallel with CMax
        VisibleVerticalClusters.add(C)

Computer Vision Layer – Stagiul 2

2.4. Se clasifica clusterele gasite la pasul anterior si se cauta feature-urile de pe teren.



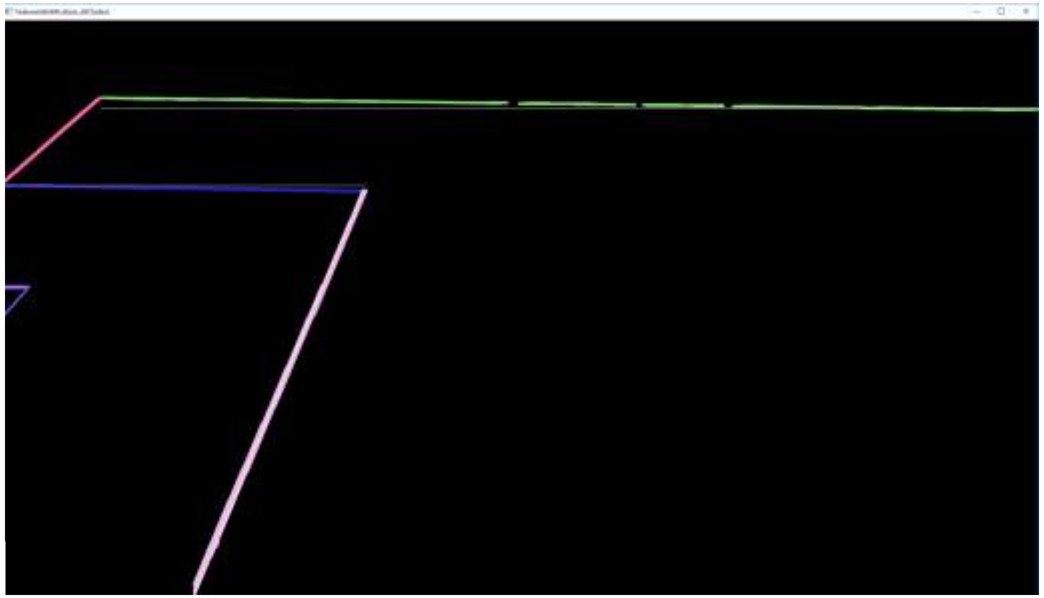


```
Step 1: try to find pitch sidelines among the visible horizonal clusters.
sidelines = {}
for the first two longest clusters HC in VisibleHorizontalClusters set
    if HC not long enough
        continue

    RS = representive segment of HC
    if one of the two RS's endpoints are in the safe part of the input image considering height ( i.e.
Y coordinate is in the image and not in the first or last 3%)
      if HC is in the upper part of image
        HC.feature = SIDELINE_UP
      else
        HC.feature = SIDELINE_DOWN

      sidelines.add(HC)

Step 2 : find endlines and midline
endlines = {}
for each VisibleVerticalCluster VC

    if VC doesn't intersects any of the sidelines
        continue
    R = intersection between VC a sideline S
    if R is not an endpoint of S
      // this is the midline. Assuming that we can't identify other features so exit
      VC.feature = MIDLINE
      if R is in the upper part:
      output.addKepoint(MIDLINE_UP)
    else
      output.addKeypoint(MIDLINE_DOWN)
      return
    else
      VC.feature = ENDLINE;
      endlines.add(VC)

    if R is in the upper part:
      output.addKepoint(ENDLINE_UP)
    else
      output.addKeypoint(ENDLINE_DOWN)

      If endlines is empty
        return
      // Not supported – usually camera sees only one sideline. Otherwise it means that it is a
      top view camera and is not safe to process in this state
      if endlines.size() > 1
        return;
      isLeftSided = true if endlines[0] is in left side of the image else false

Step 3: eliminate the false vertical clusters that are part of the ellipses around the big box or
middle line;
```

Computer Vision Layer – Stagiul 2

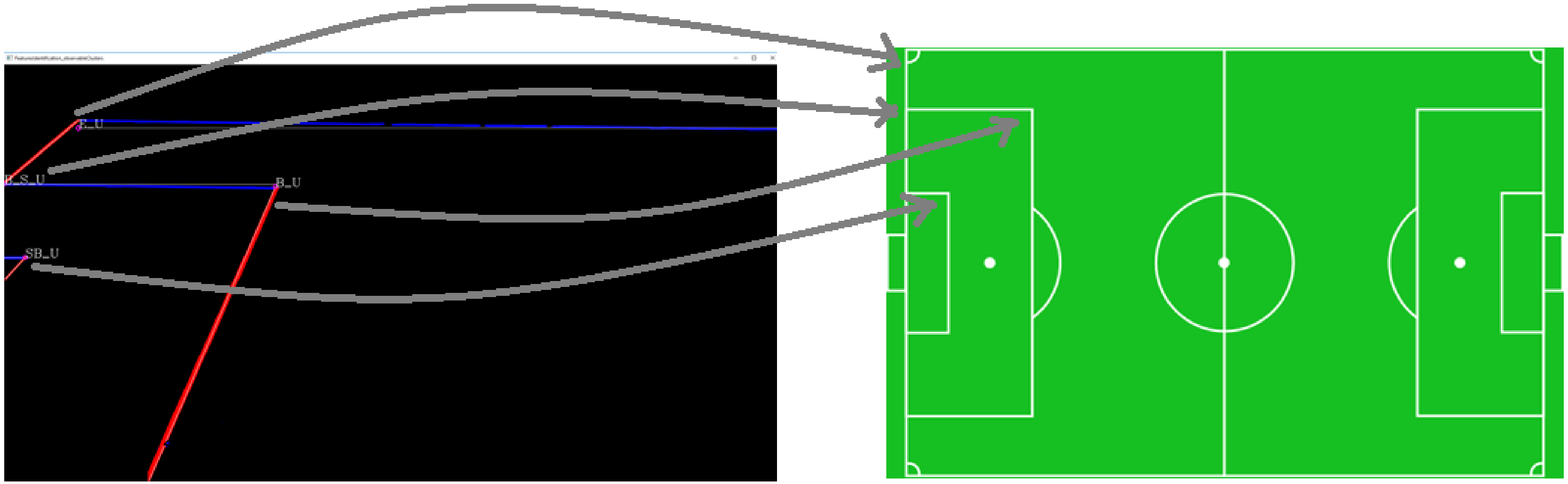
```
// Find and assume the big box cluster
// It is the longest vertical cluster which is not sideline
bigboxCluster_vertical = nil
bigboxVerticalLimit = nil
For each VisibleVerticalCluster VC ordered by length
    if VC is endline or VC is not parallel with endline
      continue

    R = VC.representativeSegment
    bigboxCluster_vertical = VC

    if isLeftSided
        bigboxVerticalLimit = min(R.start.X, R.end.X) + safeOffsetX
    else
        bigboxVerticalLimit = max(R.start.X, R.end.X) - safeOffsetX

for each VisibleVerticalCluster VC
    If (isLeftSided AND VC.minX > bigboxVerticalLimit )
         OR
       (isLeftSided == false AND VC.minX < bigboxVerticalLimit)
           Remove VC from VisibleVerticalCluster

smallBoxCluster_vertical = longest vertical cluster between endlines[0] and
bigboxCluster_vertical
```

Step 4: similar with Step 3, find bigboxCluster_horizontal[2] and smallBoxCluster_horizontal[2]
Step 5: check intersections between bigboxCluster_horizontal[2] and smallBoxCluster_horizontal[2] with endline[0]. Add the intersection points as keypoints features to the output named: BOX_ON_SIDELINE_UP / BOX_ON_SIDELINE_DOWN, respectively SMALLBOX_ON_SIDELINE_UP / SMALLBOX_ON_SIDELINE_DOWN.
Step 6: check intersections between the big box horizontal and vertical endpoints, and small boxes one. These keypoints are: BOX_UP / BOX_DOWN and SMALLBOX_UP / SMALLBOX_DOWN.



Computer Vision Layer – Stagiul 2

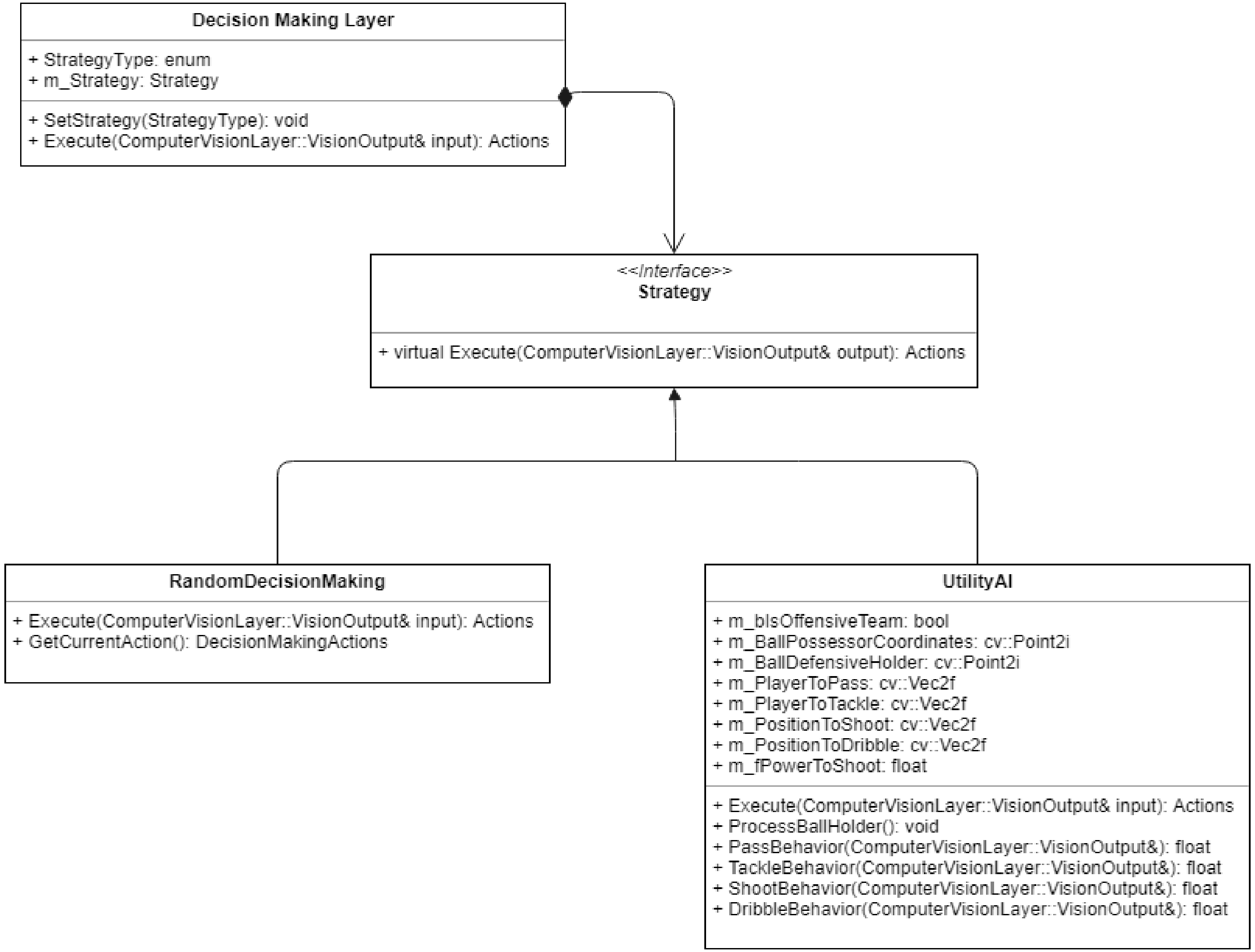
2.5. Se utilizeaza feature-urile gasite pentru a calcula matricea de homografie.



Decision Making Layer

- Primeste ca input, output-ul layer-ului de Computer Vision prelucrat in stagiul 3 al acestuia.
- Foloseste Strategy Pattern – ofera posibilitatea selectiei strategiei (algoritmului) la runtime.
- 2 strategii posibile:
    - Random
    - AIBehavior

```cpp
class DecisionMakingLayer
{
public:
    enum StrategyType
    {
        Random,
        AIBehavior,
    };

    DecisionMakingLayer() { m_Strategy = NULL; }
    void SetStrategy(int _type);
    Actions Execute(ComputerVisionLayer::VisionOutput& input);

private:
    Strategy * m_Strategy;
};
```



Decision Making Layer

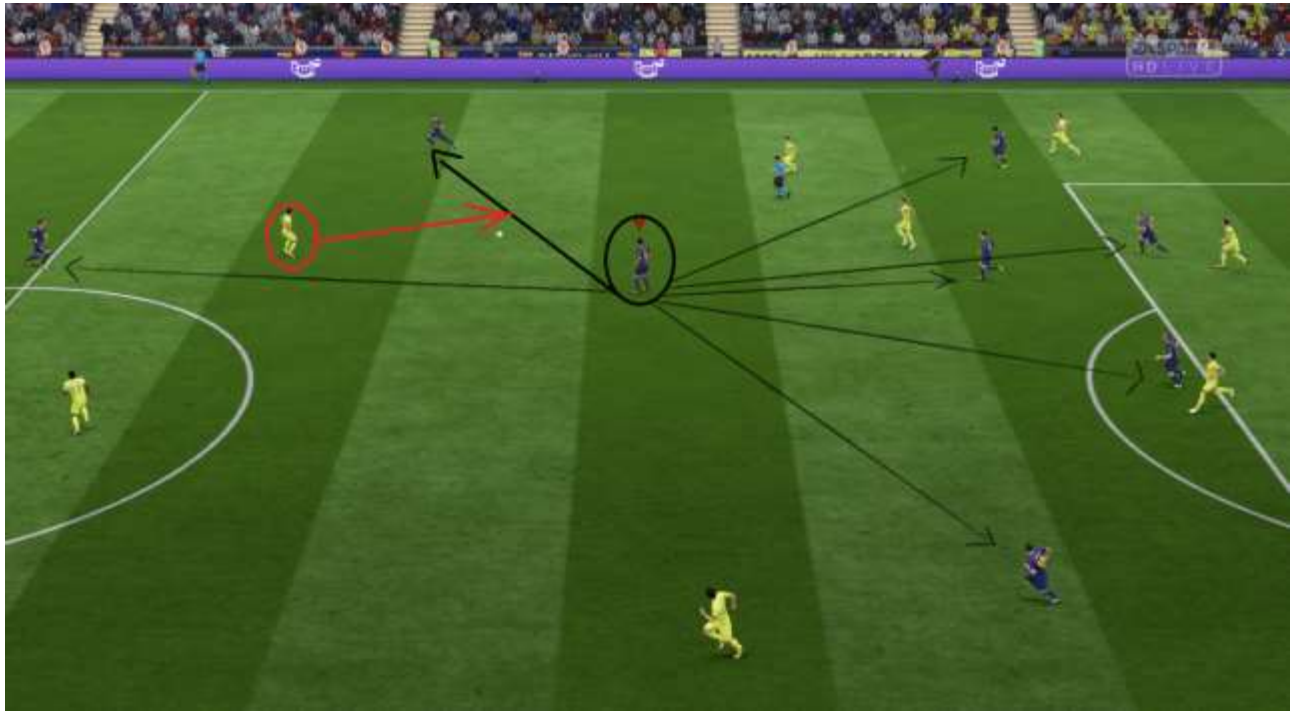
- Tipurile de actiuni folosite:

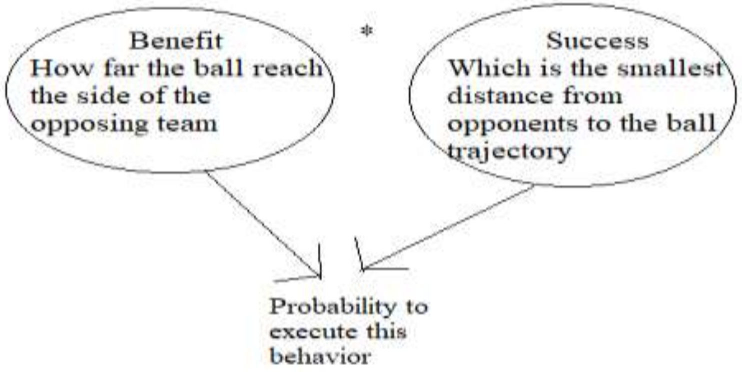
```
struct Actions
{
	float	fStickAngle;
	bool	bDribble;
	bool	bShoot;
	bool	bShootChip;
	bool	bShootDriven;
	bool	bPass;
	bool	bTackleSlide;
	bool	bTackleStand;
	bool	bSprint;
	bool	bNoAction;

	float  fPower;


	void reset()
	{
		fStickAngle = 0.0f;
		bDribble = false;
		bShoot = false;
		bShootChip = false;
		bShootDriven = false;
		bPass = false;
		bTackleSlide = false;
		bTackleStand = false;
		bSprint = false;
		bNoAction = false;

		fPower = 0.0f;
	}

	Actions()
	{
		reset();
	}
};
```

- AIBehavior are 4 tipuri de comportamente reprezentative pentru actiunile folosite:
  - Pass Behavior
  - Shot Behavior
  - Tackle Behavior
  - Dribble Behavior

Decision Making Layer

- Pass Behavior





Decision Making Layer

- Tackle Behavior

Benefit = reprezinta unghiul dintre player-ul controlat si posesorul de minge

Success = reprezinta distanta dintre player-ul controlat si posesorul de minge (in functie de distanta : stand tackle / slide tackle)

Decision Making Layer

- Shot Behavior

P = (B_up + (b_up – B_up) * 2)

P = (B_up + B_down) / 2 + Vec2(fieldSideOfAttack * SOME_PIXELS, 0);

Decision Making Layer

- Dribble Behavior

Benefit = reprezinta distanta pana la care se face dribbling

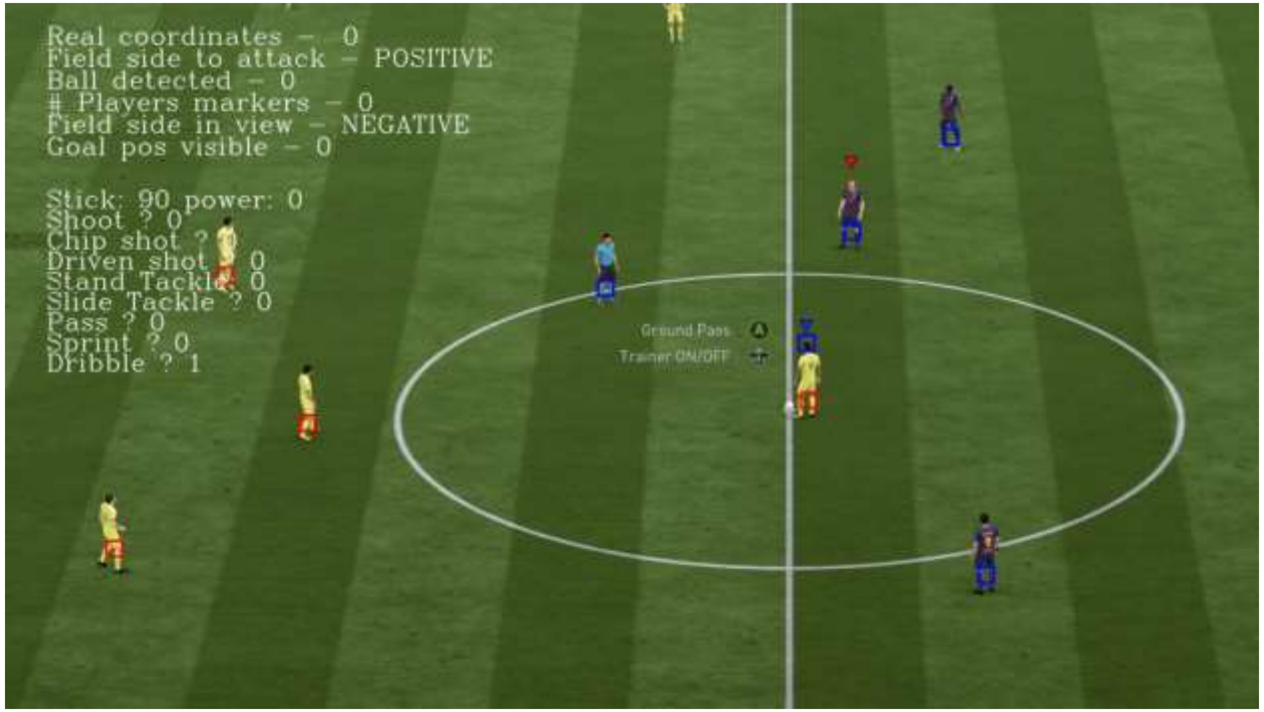
Success = reprezinta distanta minima dintre player-ul controlat si echipa adversa

Input Sending Layer

- Primeste ca input, output-ul de tipul Actions de la layer-ul de Decision Making si trimite datele catre device-ul vJoy.

Debug

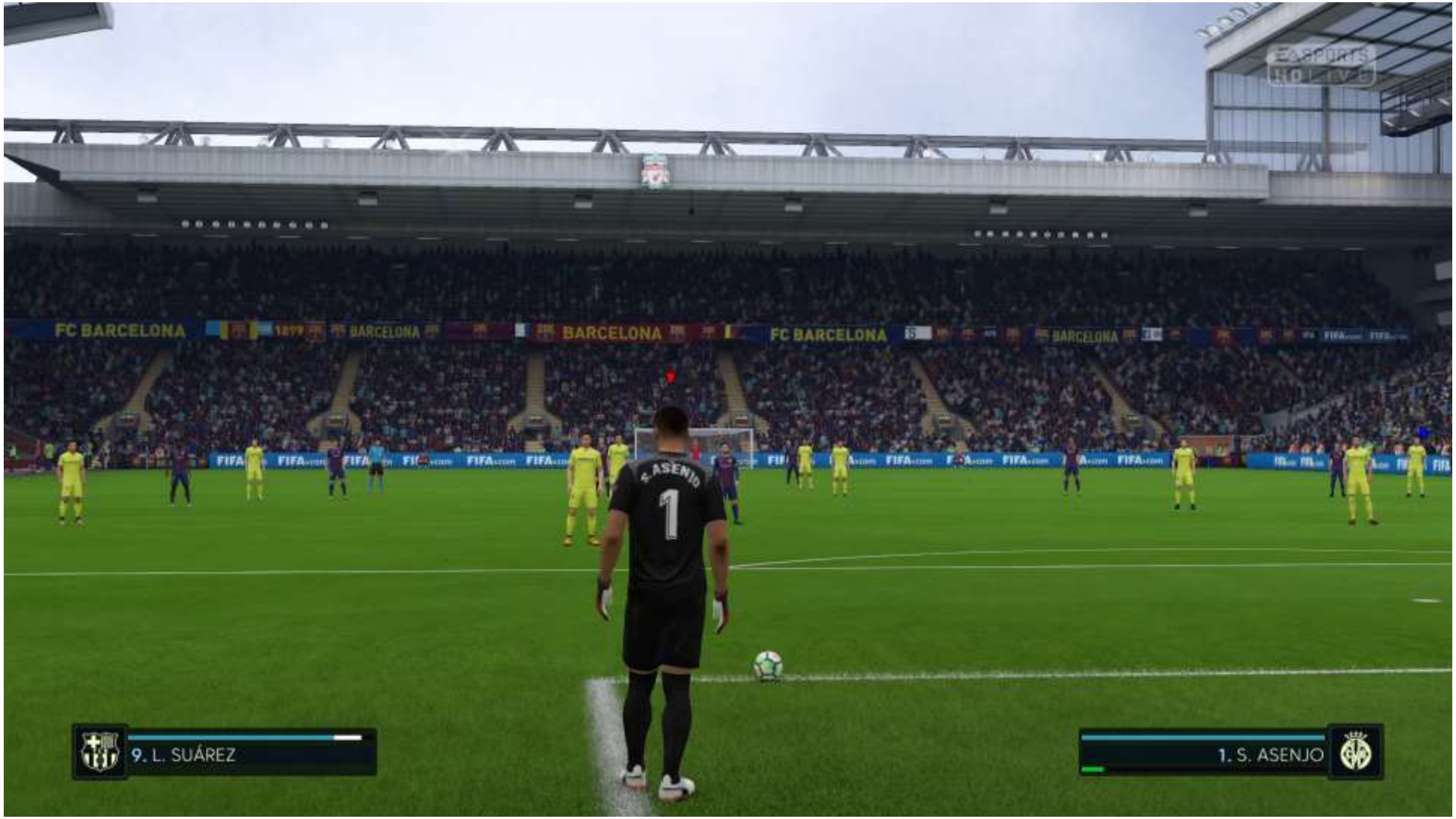
- Screenshot / custom frames



Demo

Demo

Demo



FC BARCELONA
BARCELONA
FC BARCELONA
S.ASENJO
1
9. L. SUÁREZ
1. S. ASENJO

Concluzii si dezvoltarea aplicatiei

- CORE i7 – 8700K, paralelizarea default OpenCV, utilizand 4 core-uri:
  - HOG - ~2.0 sec / imagine
  - MultiBox Detector (SSD) - ~0.8 sec / imagine
  - Algoritmul customizat – ~0.2 sec / imagine

- Detectarea keypoint-urilor – utilizarea unui algoritm mai generic – Flood Fill algorithm
- Imbunatatirea AI-ului, adaugarea unor noi actiuni si imbunatatirea celor existente
- Paralelizare GPU – imbunatatirea timpului de detectare
- Utilizarea retelelor recurente (RNN) – pentru antrenare – si pornind de la acest model invatat putem aplica retele DRL (Deep reinforcement learning).

Va multumesc!