**Hough.java**

**Constructeur**

**public Hough(int width, int height)**

**Points d’entrée :**

* width : largeur de l’image 🡪 entier
* height: hauteur de l’image 🡪 entier

**Méthodes (Transformée de Hough)**

**public void vote(int x, int y)**

**Points d’entrée :**

* x : largeur de l’image/2 🡪 entier
* y : hauteur de l’image/2 🡪 entier

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public List<double[]> getWinners(int threshold, int radius) 🡪** on récupère la valeur extreme de la transformée de Hough

**Points d’entrée:**

* threshold : seuil de l’image
* radius : rayon

**Point de sortie:**

* winners: tableau contenant les valeurs extremes de Rho et Théta 🡪 tableau de réel

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**private int distance(int r0, int t0, int r1, int t1)**

**Points d’entrée:**

* r0: point 0 de Rho 🡪 entier
* t0: point 0 de Théta 🡪 entier
* r1: point 1 de Rho 🡪 entier
* t1: point 1 de Théta 🡪 entier

**Point de sortie:**

* dist: Retourne la valeur minimale entre dist et le maximum de la valeur absolue entre (r0-r1) et (t0-t1) 🡪 entier

**Méthodes (Conversions)**

**public int RhoToIndex(double rho)**

**Point d’entrée:**

* rho : 🡪réel

**Point de sortie :**

* On retourne un entier qui est spécialement converti pour rentrer dans notre index (matrice de valeur de Rho)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**public double IndexToRho(int index)**

**Point d’entrée:**

* index : 🡪entier

**Point de sortie :**

* On retourne un reel qui vient de la conversion d’un entier (Rho) de la matrice d’index.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public int ThetaToIndex(double theta)**

**Point d’entrée:**

* theta : 🡪réel

**Point de sortie :**

* On retourne un entier qui est spécialement converti pour rentrer dans notre index (matrice de valeur de Theta)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public double IndexToTheta(int index)**

**Point d’entrée:**

* index : 🡪entier

**Point de sortie :**

* On retourne un réel qui vient de la conversion d’un entier (Theta) de la matrice d’index.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public double[] rhotheta\_to\_ab(double rho,double theta)** : conversion de rho et theta pour permettre son utilisation dans une équation de droite Y=a\*X+b

**Point d’entrée:**

* rho: reel
* theta: reel

**Point de sortie:**

* a, b: reel

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Accesseurs**

**public int getMaxIndexTheta()**

**Point de sortie :**

* maxIndexTheta: entire 🡪 on récupère la valeur maximale de theta dans l’index

**public int getMaxIndexRho()**

**Point de sortie:**

* maxIndexRho: entier 🡪 on récupère la valeur maximale de rho dans l’index

**public int[][] getAccumulator()**

**Point de sortie**

* acc: tableau 2 dimensions d’entier 🡪 on récupère ???

**PictureHandler.java**

**Constructeur**

**public PictureHandler(PhotoFragment cxt, int callerId)**

**Points d’entrée:**

* cxt: photo du parent 🡪 PhotoFragment
* callerId: entier

**Méthodes**

**public void onPictureTaken(byte[] data, Camera camera) 🡪** decode de l’image Bitmap

**Points d’entrée:**

* data: tableau d’octets
* camera: Camera

**protected** **void** **onPreExecute**() 🡪 pour chaque nouvelle ligne, on créé un nouvel HashMap les contenant

**protected** Void **doInBackground**(Bitmap... pictureFile) 🡪 dans cette méthode, on charge l’image enregistré puis on éxécute la transformée de Hough puis l’extraction des lignes de l’image.

**Points d’entrée:**

* pictureFile: type Bitmap

**protected void onPostExecute(Void result)**

**private void doTH(Bitmap img0):** application de l’algorithme de Hough sur l’image

**Points d’entrée:**

* img0: type Bitmap

**private void doLinesExtraction(Bitmap img0) :** permet de faire l’extraction des lignes suite à la transformée de Hough

**Points d’entrée:**

* img0: type Bitmap

**private void sendLinesToDrawToUiThread(HashMap<Integer, ArrayList<Point>> lines) 🡪** Permet de dessiner les lignes stockées dans le Hashmap.

**Points d’entrée:**

* lines: type HashMap<Integer, ArrayList<Point>> 🡪 Un tableau qui a comme clef des entiers permettant de retrouver plus facilement les listes de points de chaque ligne précédement stockée.