# Lukas-Kanade optical flow algorithm

**Tudorica Andrei**

**30431**

## 1. Introduction

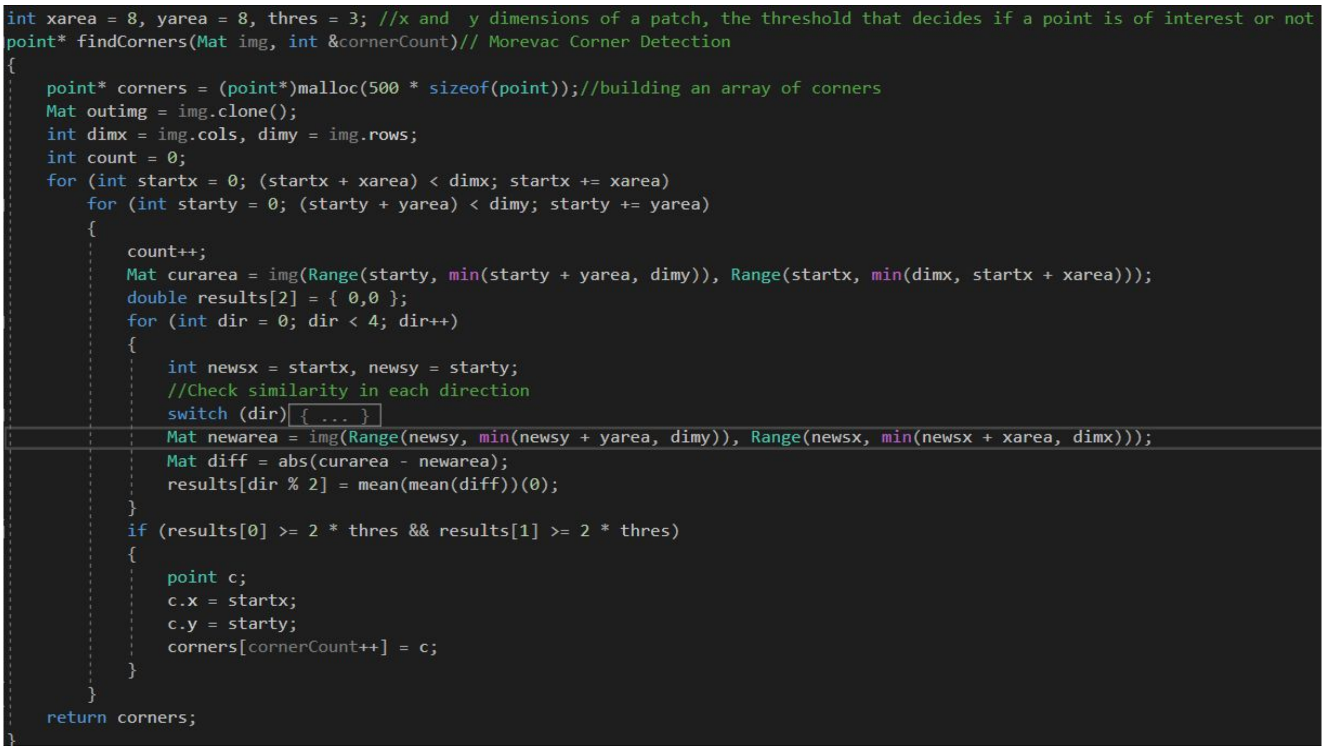
In this project I am aiming to create a c++ application to track optical flow in images captured by the webcam, using Lukas-Kanade optical flow estimation algorithm.

## 2. Related work

- https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method
- https://en.wikipedia.org/wiki/Optical_flow
- https://en.wikipedia.org/wiki/Corner_detection#The_multi-scale_Harris_operator

## 3. Proposed solution

- The chosen algorithm implies 2 big steps: Finding appropriate interest points and then applying the Lukas-Kanade optical flow algorithm.
- For the corner detection I chose the Moravec corner detection. This algorithm defines a corner as a point with low self-similarity. This means a point is considered to be a corner if it has very low similarity compared to the patches that it neighbours. The similarity is the sum of the squared differences between the corresponding pixels of two neighbouring patches.



```cpp
int xarea = 8, yarea = 8, thres = 3; //x and  y dimensions of a patch, the threshold that decides if a point is of interest or not
point* findCorners(Mat img, int &cornerCount)// Morevac Corner Detection
{
    point* corners = (point*)malloc(500 * sizeof(point));//building an array of corners
    Mat outimg = img.clone();
    int dimx = img.cols, dimy = img.rows;
    int count = 0;
    for (int startx = 0; (startx + xarea) < dimx; startx += xarea)
        for (int starty = 0; (starty + yarea) < dimy; starty += yarea)
        {
            count++;
            Mat curarea = img(Range(starty, min(starty + yarea, dimy)), Range(startx, min(dimx, startx + xarea)));
            double results[2] = { 0,0 };
            for (int dir = 0; dir < 4; dir++)
            {
                int newsx = startx, newsy = starty;
                //Check similarity in each direction
                switch (dir){ ... }
                Mat newarea = img(Range(newsy, min(newsy + yarea, dimy)), Range(newsx, min(newsx + xarea, dimx)));
                Mat diff = abs(curarea - newarea);
                results[dir % 2] = mean(mean(diff))(0);
            }
            if (results[0] >= 2 * thres && results[1] >= 2 * thres)
            {
                point c;
                c.x = startx;
                c.y = starty;
                corners[cornerCount++] = c;
            }
        }
    return corners;
}
```

- The Lukas-Kanade is a differential method for optical flow estimation. This method assumes that the displacement of the contents in an image between 2 frames is small and is almost constant in a patch around a pixel p. Every pixel satisfies the optical flow equation:

  $$I_x V_x + I_y V_y = -I_t$$

  That means the local image velocity vector must satisfy.

  $$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned}$$

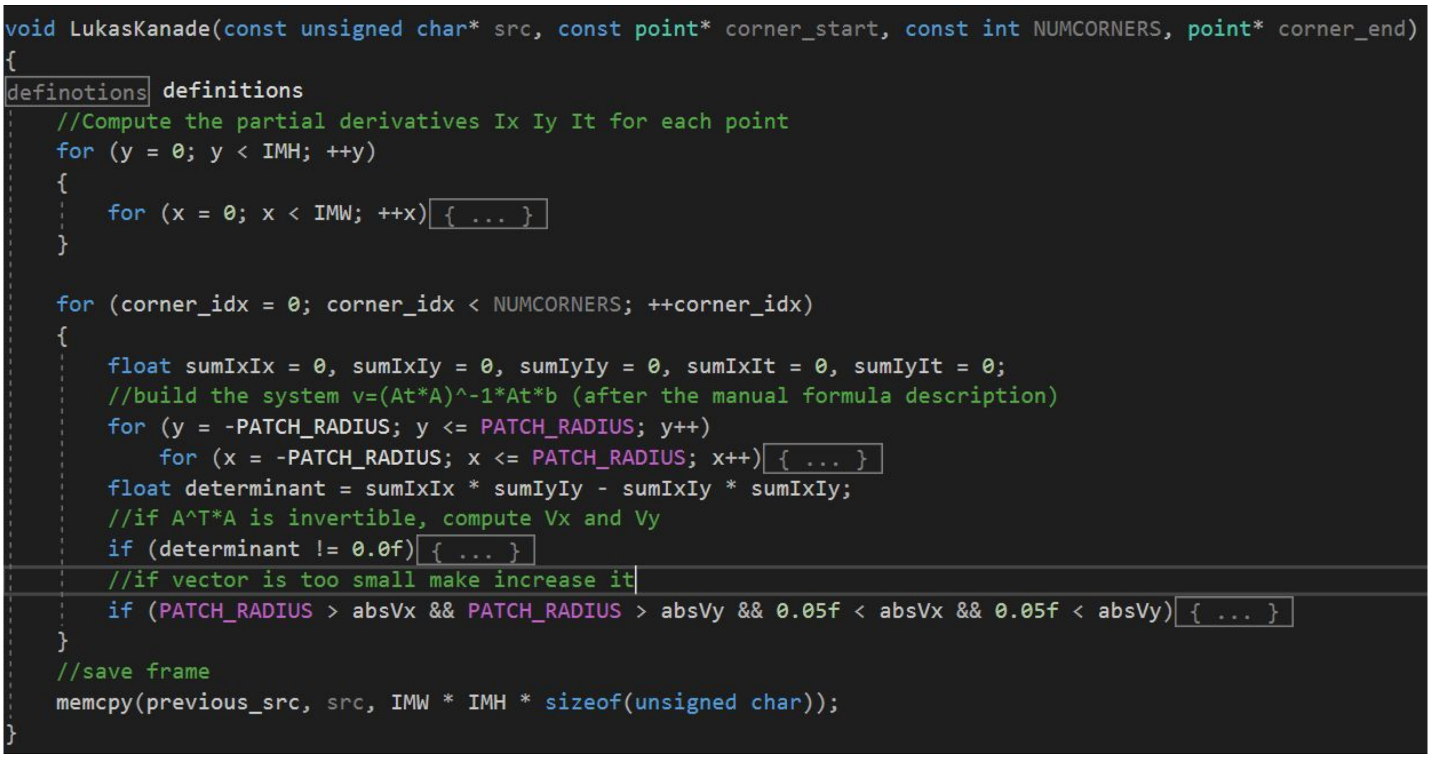
  q1, q2, q3, … , q4 - pixels
  Ix(qi), Iy(qi), It(qi) - partial derivatives of image I with respect to coordinates x, y and time t. If we write the equations in the form of matrices we obtain Av=b:

  $$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

  So the vector can be obtained by computing

  $$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

By giving I all the values from 1 to the number of points to follow (The points we selected in the previous step)



```
void LukasKanade(const unsigned char* src, const point* corner_start, const int NUMCORNERS, point* corner_end)
{
definotions definitions
    //Compute the partial derivatives Ix Iy It for each point
    for (y = 0; y < IMH; ++y)
    {
        for (x = 0; x < IMW; ++x){ ... }
    }

    for (corner_idx = 0; corner_idx < NUMCORNERS; ++corner_idx)
    {
        float sumIxIx = 0, sumIxIy = 0, sumIyIy = 0, sumIxIt = 0, sumIyIt = 0;
        //build the system v=(At*A)^-1*At*b (after the manual formula description)
        for (y = -PATCH_RADIUS; y <= PATCH_RADIUS; y++)
            for (x = -PATCH_RADIUS; x <= PATCH_RADIUS; x++){ ... }
        float determinant = sumIxIx * sumIyIy - sumIxIy * sumIxIy;
        //if A^T*A is invertible, compute Vx and Vy
        if (determinant != 0.0f){ ... }
        //if vector is too small make increase it
        if (PATCH_RADIUS > absVx && PATCH_RADIUS > absVy && 0.05f < absVx && 0.05f < absVy){ ... }
    }
    //save frame
    memcpy(previous_src, src, IMW * IMH * sizeof(unsigned char));
}
```

## 4. Experimental results

- The experiments on the application were done using the live feed from the video camera of the laptop but can be changed to a surveillance camera.
- The application was tested with objects placed close to the camera and at a medium distance.

## 5. Conclusions

- The c++ application manages to track the movement captured on the camera really well.
- The Corner detection / Interest point detection should be improved using some other Algorithm like the Förstner corner detector., because if an edge is present that is not in the direction of the neighbours (horizontal, vertical, or diagonal), then the edge will be incorrectly chosen as an interest point.

## 6. Bibliography

- **https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method**
- **http://www.cse.psu.edu/~rtc12/CSE486/lecture30.pdf**

- https://pdfs.semanticscholar.org/6841/d3368d4dfb52548cd0ed5fef29199d14c014.pdf
- https://stackoverflow.com/
- https://en.wikipedia.org/wiki/Corner_detection#Moravec_corner_detection_algorithm