1.General:

* VideoCapture.read() returns whether cv2 was able to get a frame, and the actual frame (which is a Numpy array)
* cv2.imshow(title\_of\_window, frame\_array) displays an image.
* Codul de mai jos asteapta 2 msecunde dupa ce q a fost apasat si inchide ferestrele:

if cv2.waitKey(2) & 0xFF == ord('q'):  
 break

## 2. Shrink the frame!

* Vrem imaginea cu mai putini pixeli ca sa avem o executie mai rapida
* Ca sa obtinem size ul actual:

original\_height, original\_width = frame.shape[:2]

* Ca sa facem resize: cv2.resize(frame, (new\_w, new\_h)

## 3. Convert the frame to Grayscale!

* **cv2.COLOR\_BGR2GRAY**: This is the conversion code specifying the conversion from BGR (Blue-Green-Red) color space to grayscale. In OpenCV, color images are typically represented in the BGR color space, where each pixel is represented by three values: blue, green, and red. Grayscale images, on the other hand, have only one value representing the intensity of each pixel, ranging from black (0) to white (255).
* **NTSC** – NATIONAL TELEVISION SISTEM COMMITTEE - 0.299 ∙ Red + 0.587 ∙ Green + 0.114 ∙ Blue

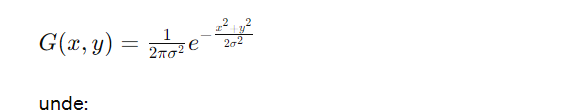
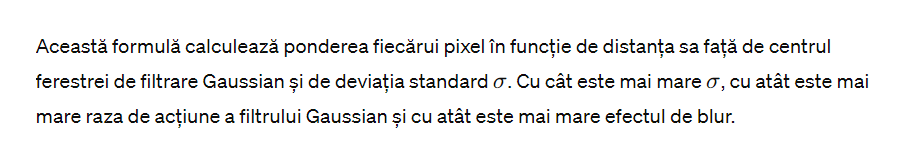
## 4. Select only the road!

* 1. Cream o frame negru de size ul frame ului init (np.array cu 0)
* 2. Cream punctele, in ordine trigonometrica, stg sus (0,0)
* Formam un array din ele, adaugam trapezul in frame ul blac, cu cv2.fillConvex(black\_frame, points, 1 – 1 de la culoare cu 0 si 1)  
  pentru a face display cv2.imashow – new\_frame \* 255 --- ca sa avem la 0 black si unde e 1 sa imultim cu 255 white
* Deci dupa ce am facut asta, avem un np.array de 0 si 1, 1 e acolo unde vrem culoare, si daca inmultim cu matricea frame anterioara, 0 va fi 0 si da negru iar 1 culoarea din anterioara

## 5. Get a top-down view! (sometimes called a birds-eye view)

* Partea de sus e mai blurata ca are mai putini pixeli si sunt intinsi ca sa acopere intreaga portiune
* Cream o matrice de transformare care primeste coordonatele frame ului de pus, coordonatele noi
* Facem strech ul cu wrapPerspective si ii dam frame ul , matricea de transformare si noile size.

## 6.Blur

* To blur an image we need to make each pixel to be the average value of its neighbors.
* Cu cati mai multi vecini cu atat mai blurata
* Folosim o matrice si la un moment dat fiecare pixel va fi in centru, si facem average ca sa bluram
* Numerele trebuie sa fie impare in asa fel in cat punctul sa fie in mijloc
* 1 \* 1 fiecare pixel ar fi chiar el, nu am blura nimic
* Am putea lua doar un pixel din cei din centru, mereu acelasi
* Au un alt efect de blur deoarece au mai putini pixeli din care sa faca media, cum am putea rezolva, fie putem padda in oglinda, sau aplicam un blur mai mic cat se poate
* Alte tipuri de blur:
* Există mai multe metode de blur, fiecare cu avantaje și utilizări specifice. Iată câteva dintre ele:
* Gaussian Blur: Aceasta este una dintre cele mai comune metode de blur. Se bazează pe aplicarea unui filtru Gaussian pentru a diminua claritatea imaginii. Gaussian Blur este eficient în reducerea zgomotului și a detaliilor nedorite, făcând imaginea mai uniformă.
* 
* 

## 7.Sobel

* CUM SE APLICA UN FILTRU:
  + - Avem o matrice a filtrului, ne miscam in matricea de pixel in asa fel in care fiecare pixel sa fie in centru, inmultim fiecare element cu elementul corespunzator in matrice, insumam si punem in centru
    - Cum pe margine nu avem suficienti pixeli, aceste elemente nu vor ajunge vreodata in centru si vom avea o imagine putin mai mica
* Ce se intampla defapt in filtrarea sobel:
  + - Cand mergem cu matricea peste pixeli, daca pixelul din dreapta si cel din stanga au aceeasi valoare (ca si culoare) acestia isi anuleaza valoarea iar cel din mijloc primeste 0, adica black(idem vertical cu sus jos) – stg, + dr, - sus, + jos
    - Daca exista o schimbare drastica de culoare , va fi luminata cu alb cel din mijloc
    - Obtinem 2 matrici pt orizontal si verical si le inmultim cu m = rad(m1\*\*2m2\*\*2)
* Intrebari de pe discord:
  + Cum functioneaza filtrele:
    - Matrici care se misca peste matricea principala, avand un element in mijloc si se inmultesc vecinii cu poz coresp in matricea filtrului sia corda valoarea celui din centru
  + Ce as putea sa fac sa aplic filtrul doar pe jumatate de ecran?
    - Merg cu matricea doar pana la jumatate, sau fac o copie de frame pana la width//2, aplic filtrul si creez o noua matrice cu pixelii in prima jum de la aceasta noua matrice si pe cealalta de la anterioara
  + Ce se intampla daca nu foloseam trapezul ca masca?
    - Luam elemente din imagine ce nu ne ar fi interesat, obtineam dupa aplicarea filtrului probabil mai multe puncte albe, iar regresia nu ar mai fi dat corect sa tina banda

## 8. Binarized the frame:

* O valoare threashould, ce e sub e negru ce e peste e alb
* Intrebari de pe discord:
  + Threshold: ce face argumentul 3, ce se intampla daca il schimbi, alte metode de a face threshold?
    - Deci avem matricea frame ului, valoare dupa care se inpart in 2, ce e peste ce e sub, ce valoare maxima iau, gen 255 vine de la alb si ultimul ne arata ca impartim in 2
* **cv2.THRESH\_BINARY\_INV**: Această metodă este similară cu **cv2.THRESH\_BINARY**, dar inversează rezultatul, adică pixelii care sunt mai mari decât pragul vor fi setați la 0, iar cei care sunt mai mici sau egali cu pragul vor fi setați la valoarea maximă specificată.
* **cv2.THRESH\_TRUNC**: În această metodă, pixelii care sunt mai mari decât pragul vor fi setați la valoarea pragului, iar ceilalți pixeli vor rămâne la valorile lor originale.
* **cv2.THRESH\_TOZERO**: În această metodă, pixelii care sunt mai mici decât pragul vor fi setați la 0, iar ceilalți pixeli vor rămâne la valorile lor originale.
* **cv2.THRESH\_TOZERO\_INV**: Similar cu **cv2.THRESH\_TOZERO**, dar inversează rezultatul, adică pixelii care sunt mai mici decât pragul vor fi setați la 0, iar ceilalți pixeli vor rămâne la valorile lor originale.
  + - Iar de mana putem itera si verifica care e mai mare care e mai mic si acordam valori

## 9. Impartirea in 2:

* Impartim matrice pe width in 2, acordam primelor 0.05 coloane black
* Apoi pt fiecare parte folosim argwhere(pixel ==1) care retuneaza pe fiecare linia coordonatele y, x, adica linie coloana, (vector de vectori)
* Intrebari de pe discord:
  + ce returneaza functia argwhere, si ce tip de vector returneaza?
    - Returneaza un vector de vectori, o matrice sa zicem, unde fiecare vector (adica linie) contine un linia si coloana adica(y, x) a puctelor unde avem 1

## 10. Desenare liniilor:

* Avem coordonatele punctelor din stanga si din dreapta care sunt albe
* Functia polinomial ne da rezultatul polinomului de gr 1, y =ax+b, dar ne da b si apoi a
* Acum noi trebuie sa aflam punctul de inceput si punctul de start in fiecare frame a linie, sus y va fi 0, jos va fi inaltimea, si calculam x si tragem linie, asemenea si in dreapta
* Intrebari discord:
  + de ce mi se buguieste incadrarea de linie pe banda?
    - Apar diferiti pixeli alb, intra benzile din una in alta, regresia nu mai e buna, na da altceva
    - modifica linie de cod ex: culorile benzilor, pune banda cu 50px mai sus, culaorea usor de modificat, insa cu 50px mai sus? 😊)

## 11. Final view:

* Facem un blank frame
* Desenam doar linia din stg folosind punctele descoperite
* Acum schimbam perspectiva si ce aveam in intregul frame il mapam doar in trapezoid
* Si luam punctele acum
* La fel in dreapte
* In frame ul final desenam linii in functie de punctele alea

Partea 2

Clasa object socket:

3 atribute default: size ul hederului ce se trimite cu datele, timeout in secunde care reprezinta nr maxim de secunde cat se asteapta un raspuns, size of chunck data, un bloc, o bucata din datele trimise.

Avem doua parti:

Senderul si receive ul

1. Senderul , clasa are ca atribute, adresa ip, port ul la care se face bind, socket ul, conexiunii tcp, connexiunea stabilita, flaguri pt cand se asteapta conexiuni si se trimit mesaje.
2. Send(obj)

* este serializat într-o reprezentare de byți
* se impart datele in blocuri de lungime specifice clasei
* data\_size\_encoded este o secvență de bytes care reprezintă dimensiunea datelor serializate și este trimisă inițial pentru a indica receptorului câți bytes să aștepte înainte de a citi datele efective.
* data conține datele efective serializate, care sunt trimise după dimensiunea codificată.
* De ce sendAll() --- ca este mai sigur, se asigura ca toate datele sunt trimise pe canal si nu exista omisiuni, si exista in spate un algoritm de fragmentare nu trebuie sa implementam logica fragmentarii noi, datele sunt transmise in ordine!!!
* **send(data)**:
  + Metoda **send(data)** este utilizată pentru a trimite datele din bufferul de date specificat.
  + Dacă este posibil, aceasta va trimite toate datele specificate într-o singură tranzacție.
  + Dacă nu poate trimite toate datele odată, **send()** va returna numărul de bytes trimiși cu succes și va păstra restul datelor în buffer pentru trimitere ulterioară.
* **sendall(data)**:
  + Metoda **sendall(data)** este utilizată pentru a trimite toate datele din bufferul de date specificat.
  + Dacă nu poate trimite toate datele într-o singură tranzacție, **sendall()** va continua să trimită datele rămase până când toate datele sunt trimise complet sau o eroare apare.
  + **sendall()** asigură că toate datele sunt trimise, chiar dacă este necesar să împartă datele în pachete mai mici și să trimită fiecare pachet individual.

1. Receive obj:

* Are ca atribute o adresa ip, un port, conexiunea,adica socket ul si flaguri cat asteapta sa primeasa si cand primeste
* Metoda connect to sender se conecteaza cu sender ul
* Close inchide
* Is connected, verifica conexiunea
* Recv\_object, primese obiectul de la sender, prima data nr de niti ce ii are de descarat si dupa elementul in sine, receiiv\_all, primeste date pana cand ajunge la nr de biti pe care trebuie sa ii citeasca
* Cu recv simplu nu ne putem asigura ca toate datele sunt primite intr un pachet se pot pierde, cu receive all asteptam sa primim toate pana trecem mai departe
* De ce nu exista default, pt ca cel ce trimite poate gestiona lungimea datelor, insa cel ce primeste nu, poate ne dormim o alta logica de reasablare a datelor
* Receive cu time out: asteptam datele o perioada de timp
* Se folosește **select.select()** pentru a monitoriza starea conexiunii de rețea și pentru a aștepta pentru datele de intrare timp de **timeout\_s** secunde. Parametrul **timeout\_s** specifică timpul maxim de așteptare pentru datele de intrare.
* **select.select([self.conn], [], [], timeout\_s)** așteaptă ca **self.conn** (socket-ul conectat) să devină disponibil pentru citire timp de **timeout\_s** secunde. Dacă socket-ul devine disponibil înainte ca timpul să expire, se va întoarce o listă de socket-uri disponibile pentru citire (**rlist**).
* Dacă **rlist** nu este gol, înseamnă că socket-ul este disponibil pentru citire și există date disponibile pentru recepție.
* În acest caz, se utilizează metoda **recv(n\_bytes)** pentru a primi **n\_bytes** de date de la socket. Aceste date sunt stocate în variabila **data**.
* Funcția returnează **data**, reprezentând datele primite ca șir de bytes.
* Dacă **rlist** este gol, înseamnă că timeout-ul a expirat și nu au fost primite date în intervalul specificat de timp. În acest caz, funcția returnează **None** pentru a indica că nu au fost primite date în timp util.
* Se primește un parametru **n\_bytes**, care specifică numărul total de bytes pe care trebuie să-i primim.
* Exprimarea parametrului **timeout\_s** este opțională și specifică timpul maxim de așteptare pentru primirea tuturor datelor. Dacă nu se specifică, se va folosi valoarea implicită **ObjectSocketParams.DEFAULT\_TIMEOUT\_S**.
* Se inițializează o listă goală **data** pentru a stoca datele primite.
* Se utilizează o buclă **while** pentru a continua recepționarea datelor până când toți cei **n\_bytes** sunt primiți sau până când apare un timeout.
* În fiecare iterație a buclei, se calculează dimensiunea dorită a fragmentului de date pe care trebuie să-l primim. Aceasta este determinată de alegerea dintre dimensiunea fragmentului dorit și numărul de bytes rămași de primire.
* Se folosește metoda **\_recv\_with\_timeout()** pentru a primi fragmentul de date cu dimensiunea specificată.
* Dacă fragmentul este diferit de **None**, înseamnă că au fost primite date și sunt adăugate la lista **data**. Apoi, numărul de bytes rămași de primire este actualizat.
* Dacă fragmentul este **None**, înseamnă că a apărut un timeout și nu au mai fost primite date în timp util. În acest caz, se aruncă o excepție de tip **socket.error** pentru a indica că nu au fost primite toate datele înainte ca timeout-ul să expire.
* La final, toate fragmentele primite sunt concatenate într-un singur șir de bytes folosind metoda **b''.join(data)** și acesta este returnat.

1. Funcția apelează **\_recv\_all()** pentru a primi exact numărul de bytes specificat în **ObjectSocketParams.OBJECT\_HEADER\_SIZE\_BYTES**. Acest lucru reprezintă dimensiunea antetului (header-ului) care conține informații despre dimensiunea obiectului.
2. Dimensiunea antetului este citită din socket și stocată în variabila **data**.
3. Apoi, dimensiunea obiectului este extrasă din datele primite utilizând metoda **int.from\_bytes()**. Parametrul **'little'** indică faptul că datele sunt interpretate ca fiind în format little-endian (mai întâlnit în rețelele de calculatoare).
4. Dimensiunea obiectului este returnată ca rezultat al funcției.

În cazul în care apare un timeout în timpul primirii dimensiunii obiectului (antetului), funcția va arunca o excepție de tip **socket.error**, indicând că nu a fost posibilă primirea antetului înainte ca timeout-ul să expire.

În concluzie, **\_recv\_object\_size()** este o funcție auxiliară utilizată în procesul de primire a unui obiect prin socket, care asigură că dimensiunea obiectului este corect primită și interpretată înainte de a continua cu recepția efectivă a obiectului.

Top of Form

**recvall():** O implementare pentru **recvall()** ar trebui să primească și să adune toate datele primite prin socket până când un anumit număr de bytes a fost primit sau până când o condiție specificată a fost îndeplinită (cum ar fi un anumit byte de terminare). Totuși, implementarea unei funcții **recvall()** poate fi mai dificilă decât **sendall()**, deoarece trebuie să gestioneze bufferizarea datelor primite și să țină cont de limitările dimensiunii bufferului, posibilele întârzieri de rețea și alți factori care pot afecta recepționarea datelor.