



# opticalFlow

- At the end of the last semester, we had a problem with the region of interest. The ROI was creating problems for the opticalFlow. Sinem fixed that problem by making the ROI a rectangle instead of an odd shape.
- Then she started to work on a clustering algorithm. First, she researched about the different clustering types to find the most fitting one. She decided that density based was the most suitable option.
- She first tried a few libraries, but they didn't work the way she wanted. So, she decided to write algorithm of her own.

# Clustering

- First, she created a 2D array same dimensions as the ROI she created. Then she added every moving pixel that detected by optical flow to this array.
- Then because traversing the whole array is too slow. She decided that traversing as 100 steps is much more efficient. She checked if this 100 steps have more than x amount of moving pixels, this means there is a moving object there.
- Then she tried to connect close steps to find the vehicle that is moving and then she calculated the center of these moving steps.
- She also sent these centers to big demo code.

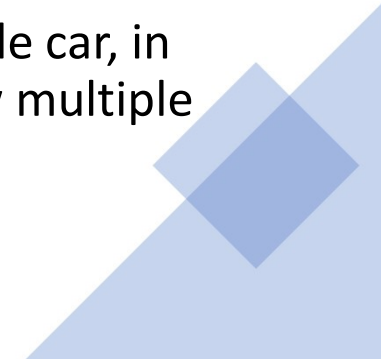


# User Interface

- In the beginning of the semester Selin researched about the different user interface libraries that can be used with Python.
- She tried to create a few demos with different libraries and decide to use pyQT5 for this project.
- She first added a truck image and created regions around it. Then she tried to add a car image, but she couldn't find a way to move images. So she decided to create rectangles which she is sable to move.
- She also changes the color of the car rectangles if they enter the danger zone (the regions around the truck).



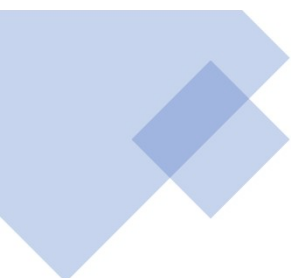

## Demo

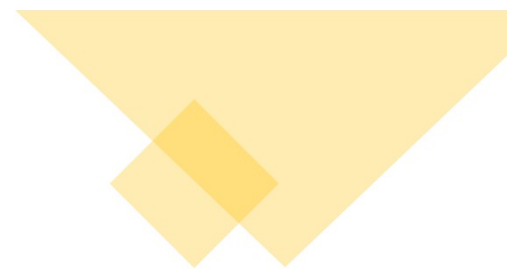
- Then, Selin created a test array to show the movement of the car rectangle.
  - When this step was successful Selin and Sinem started to combine their code. (the user interface and the opticalFlow)
  - We created a big demo code to call each other codes in one single place.
  - First, Sinem changed her code that can return x,y coordinates to the demo. Selin also changed her code that can receive this x,y coordinate from the demo code.
  - After we successfully show the movement of a single car, in real time, in the user interface, we decided to show multiple cars.
- 



## Demo (Cont.)

- Showing multiple cars was a different process than single car. So we need to change our approach completely.
- To show the single car we created a rectangle and called a move function for this rectangle. However this approach was almost impossible for multiple cars. Because the code should decide if the x,y coordinate is belonging to a new car or it's a car that previously created.
- So, we decided to create a new car object and delete the previous one for each frame of the video.
- However, we had to create the object and delete it after we used. This created a bug that took a while to fix.

- 
- We were creating an object multiple times. We first debugged it, but the code somehow was skipping the bugged part.
  - Then we change our approach and found the bugged part and fixed it. We think that we couldn't find it with debugging tool because when the code stops the Python does a garbage collection and delete the old object before creating a new one. However, when we run it regularly the compiler don't have this time and the code crushes.
  - Now we have a demo that can show multiple car in real time.
- 



Demo (Cont.)

## Demo (Cont.)

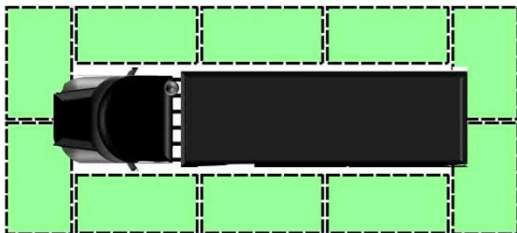
- We have two different user interfaces for front and side view.



Interface with data > InterfaceFront.py

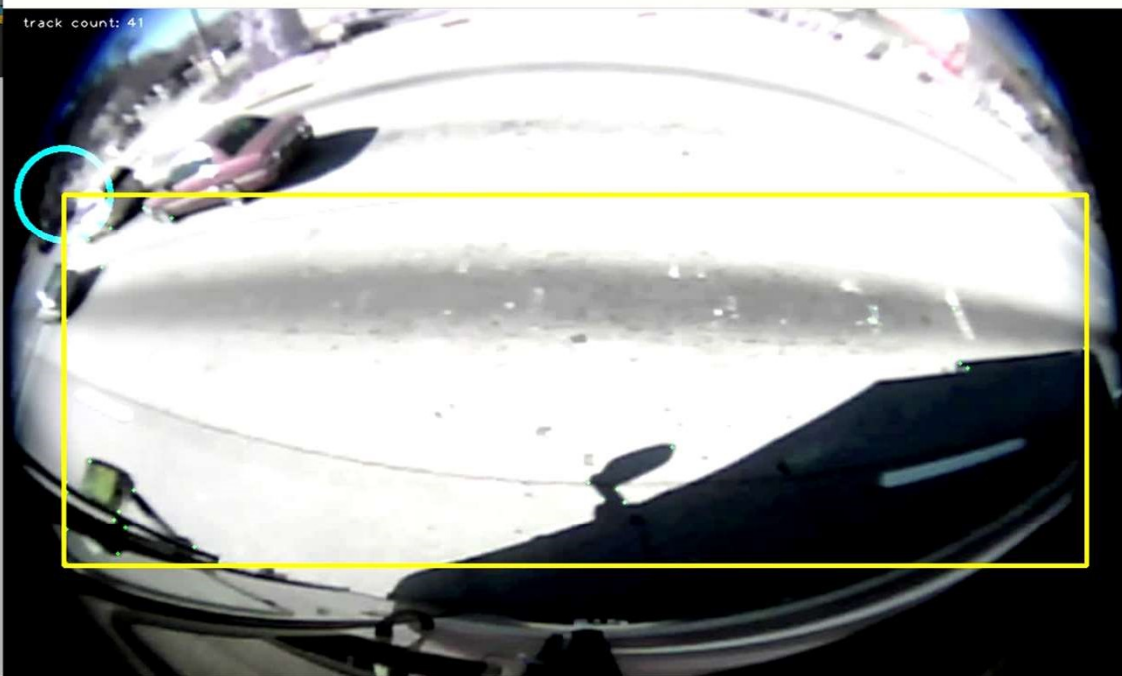
Project ▾

Truck Interface



Sketcher ROI

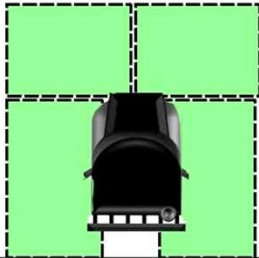
track count: 41



```
self.height = 800
```

```
self.flag=200
```

Truck Interface



```
self.rectObjArray.clear()
```

Sketcher ROI

