Summer Internship Report

Arjun Chauhan

Department of Electronics and Communication Engineering
Manipal Institute of Technology

Under the Guidance of
Mr Ferhat Ölgün

Karel Electronics

July 2019

Abstract

The automobile world is changing at a rapid pace with improvements in mileage, speed, automation and reliability. To aid this development and optimize the process further, an imaging system, which would be installed on board a car, was developed. This system would use cameras which would replace existing rear-view mirrors. The project was developed as a part of the Advanced Driver Assistance System(ADAS) being developed by Karel Electronics.For the project, an algorithm had to be developed and implemented which would detect oncoming traffic using the rear view cameras. The entire project was aimed at implementing and testing algorithms on a host PC and then porting them to an embedded system which would thereafter be used in automobiles.

Table of contents

List of figures

1. *Karel Electronics*

1.1 About Karel

Karel Electronics entered the market primarily as a communications service provider creating a telephone exchange market in Turkey in 1986 using its original designs. It provided PBX (Private Branch Exchange, a specialized telephone system in a corporation or institution) systems to users and emerged as a pioneer in Turkey's transition to electronic systems in communications. The company is established with only domestic capital and has one of the most advanced Research and Development department in communication electronics field which serves in international standards. It has its own research and development programme further diversifying into cloud solutions, security solutions, defense industry and telecommunications.

1.2 Location

Head Office

- **Address**: Kore Şehitleri Caddesi Yzb. Kaya Aldoğan Sokak 16 Zincirlikuyu 34394 İstanbul
- **Phone No.**: +902122883100

Research and Development Center

- **Address**: Cyberpark Cyber plaza B Blok Kat 3 Bilkent 06800 Ankara
- **Phone No.**:+903122650297

Production Center

- **Address**: Organize Sanayi Bölgesi Gazneliler Caddesi 10 Sincan 06935 Ankara

- **Phone No.**:+903122670244

1.3 Profile of the Company and Areas of Interest

At present, Karel is the market leader in developing and business of its communication systems and amongst the top 15 manufacturers of the world. its products are sold in more than 30 countries worldwide.

Today more than 650000 businesses are using Karel PBXs. In our country, over 50% of the business communication traffic is done via Karel systems according to the reports of all international and national research institutes and the company ranks amongst the top 500 industrial organizations in Turkey.

Karel develops and produces wired or wireless communication systems with different specialties. Major products are specialized PBXs with small, middle and large capacities and peripherals to these PBXs, IP call centers and etc. in the superstructure class; and rural PBXs and interconnection PBXs in the infrastructure class. Lately, the company enhanced its activities with Unified Communications Solutions (integrating the software that supports synchronous and asynchronous communication to provide the end user easy access to all tools from whatever computing device one is using). The company as stated earlier is also involved in development and sales in the visual communication and security systems field and in the intruder alarm systems market. In its sales, company uses authorized distributors who are present both domestically and abroad. Karel imports some complementary products for its activities in the telecommunications and also performs turn-key projects for the public and the private sector organizations by cooperating with some foreign companies. When we examine Karel as an electronic systems manufacturer, it mainly operates in the household appliances, electronics design and production in relation with this field of activity.

1.3.1 Production

The company performs its manufacturing activities in Ayas Industrial Zone in Ankara, by the coordination of its Production Planning and Control, Purchasing and Production Departments, in its plant that sits on 20,000 m^2 of area of which 10,000 m^2 is closed.

Insertion and test machines, employing the latest technology are deployed in the production line, which is under continuous development. Automatic insertion is based on axial, radial and SMD technologies. In-circuit testers and functional testers are used for testing and verification. Total automated component insertion rate of these production lines is 210,000 components per hour.

1.3.2 Sales and Marketing

Karel is the leading company in communication industry in Turkey. Karel is also the leader in PBX market with its 50% market share according to the report of MZA, the international market research institution. Moreover, it is placed first in the "Top 500 Information Technologies Companies" report prepared by Interpromedya in terms of net sales income of companies operating in the information technologies industry in Turkey in 2010. Karel is in the top 20 information technologies companies with the best performance in terms of the sales incomes for the last 3 years and in the top 10 companies according to the equipment export income. It is specified as the most valuable PBX brand according to the market research performed by the Recon Research Institute all over Turkey.

Karel, today, exports products and technologies with the Karel brand to more than 30 countries. International sales are directed by 2 regional departments located in Ankara. Local distributions are managed by more than 300 authorized dealers and more than 500 sales outlets, which are coordinated through Karel offices in İstanbul, Ankara, İzmir, Antalya and Van.

2. *Algorithms*

2.1 The Objective

The main objective of the project was to develop an algorithm capable of detecting oncoming traffic using the rear view cameras. This was primarily being developed for the automobile sector and was to be used as a driver assistance system, helping the driver get a better sense of the surroundings and help be more aware.

This car detection system is planned to be implemented on a embedded system. This objective presented itself with challenging constraints in-terms of processing power, constraints of libraries being used and overall integration. Therefore the end goal was to use a suitable algorithm for the objective while making it efficient and powerful in lieu of the constraints faced in terms of data,processing power and integration.

After evaluating the available algorithms, the most suitable algorithms would be implemented on a host PC and evaluated based on performance and speed after which these would be ported to the embedded system.

2.2 Understanding Existing Algorithms

In order to get optimal performance from the board, several algorithms and pre-existing papers were evaluated for ideas and inspiration to develop a suitable approach. Each of these papers and technologies presented a few different approaches which were suited optimally for their purpose. The ideas and their details of a few are discussed in the subsequent sections.

2.2.1 Haar Cascades

Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The algorithm has 4 stages namely

- Haar Feature Selection.
- Creating Integral Images.
- Adaboost Training.
- Cascading Classifiers.

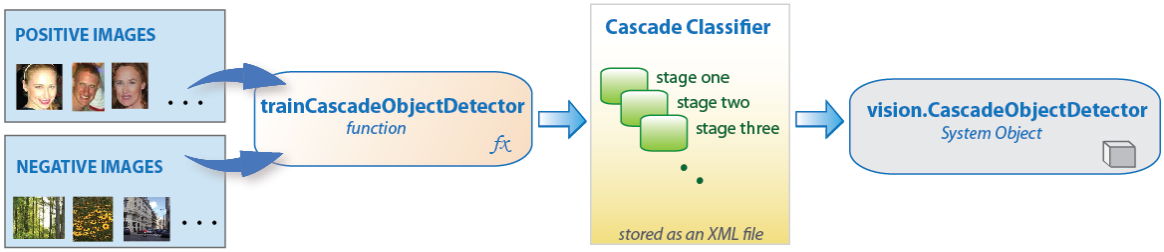

Fig. 2.1 Haar Cascade Pipeline

To train the classifier algorithm there is a requirement for obtaining *positive images*(Images containing cars) and *negative images*(Images not containing cars).Following this step,features need to be extracted from it.

The first step is extracting Haar Features. A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. To further speed up this process, Integral Images are used.

However, the set of features extracted using these filters may not always yield useful results. To take this limitation into account, an algorithm called AdaBoost (Adaptive Boosting) is used. Using this process, the algorithm selects the best useful features required to train to the classifier. AdaBoost creates these strong features by linearly combining various weaker features by assigning them a weight.

The cascade classifier consists of a collection of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called decision stumps. Each stage is trained using a technique called boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners.

Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative

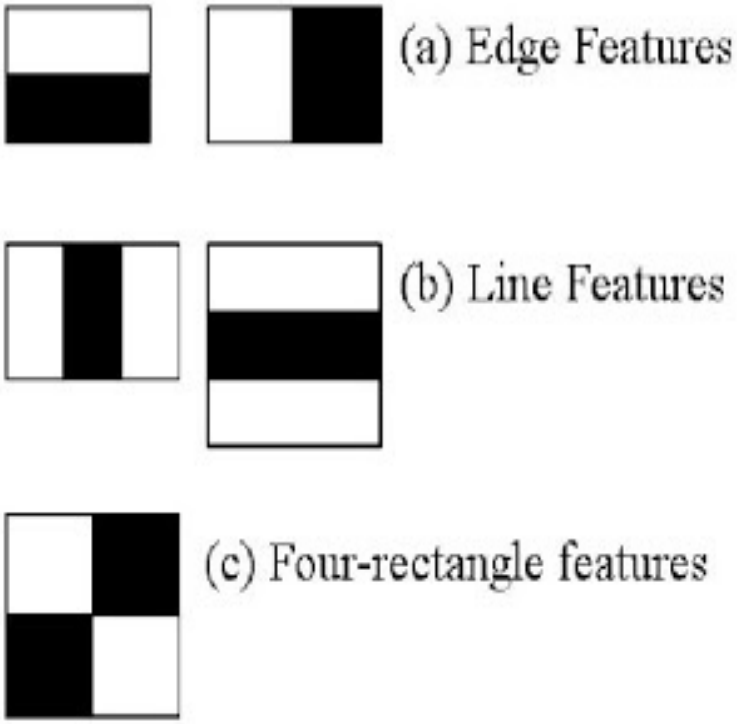

Fig. 2.2 Haar Feature Extraction Kernels

indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

However, this algorithm would not work with an eccentric or an unusual build for a car. The limitations of the Haar Cascades and improvements in bottleneck are explained in detail in **Section 5 - Future Development.**

2.2.2 Optical Vector Flow

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. The concept of optical flow was introduced by the American psychologist James J. Gibson in the 1940s to describe the visual stimulus provided to animals moving through the world. We used the Lucas Kanade method for Sparse Optical Flow. The Lucas-Kanade optical flow algorithm is a simple technique which can provide an estimate of the movement of interesting features in successive images of a scene. We would like to associate a movement vector (u,v) to every

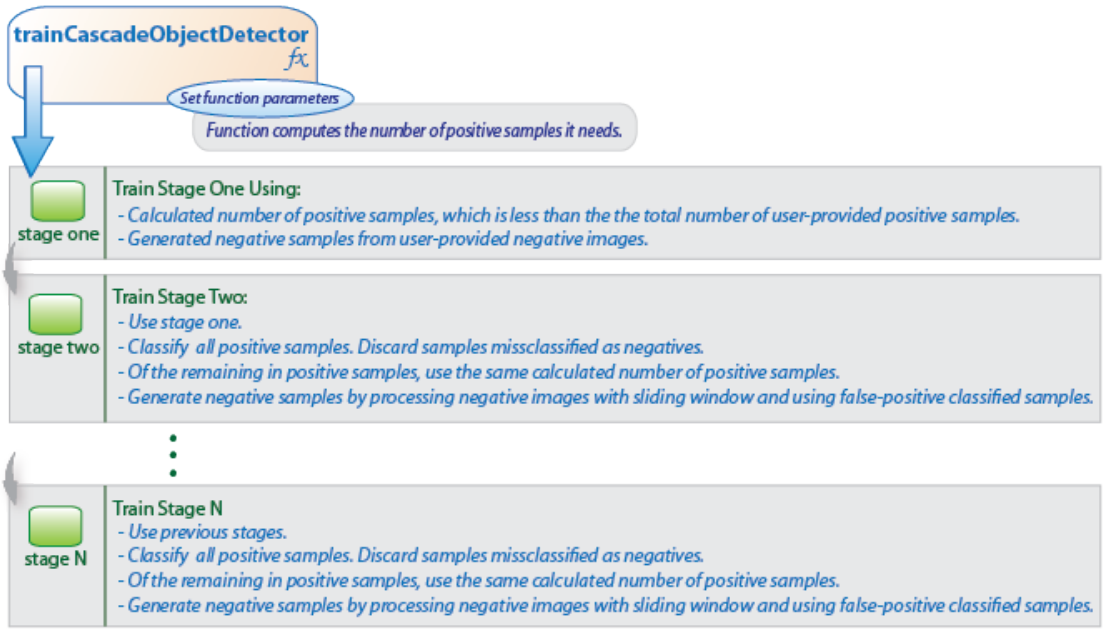

Fig. 2.3 Haar Feature Extraction Kernels

such ”interesting” pixel in the scene, obtained by comparing the two consecutive images. These "interesting" pixels were decided from taking small corner points on the ground truth image.

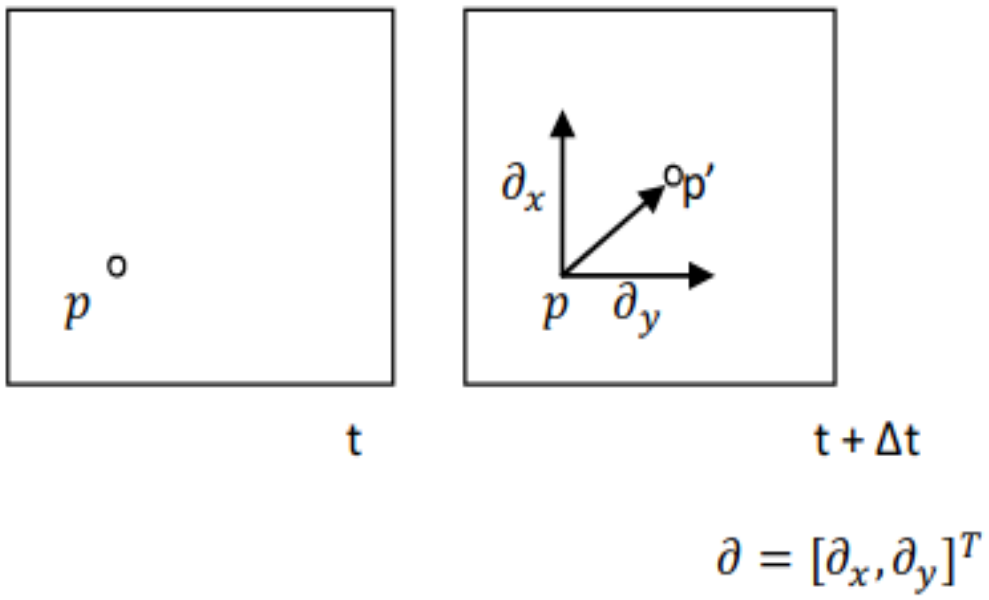

Fig. 2.4 Optical Flow Vectors

However, despite being fast in its implementation, it was not chosen for the final implementation because of the inherent assumptions which are made for this kind algorithm. These assumptions are stated below

- The images depict a natural scene containing textured objects exhibiting shades of gray (different intensity levels) which change smoothly. (i.e. algorithm is very sensitive to lighting)
- The two consecutive images in a video sequence are separated by a small time increment Δ t in such a way that objects have not displaced significantly. (i.e. algorithm works best with slow moving objects

2.2.3 Kernalized Correlation Feature(KCF) Tracker

Tracking is defined as the problem of generating an inference about the motion of an object given a sequence of images. Given a patch of an image, a tracker keeps "track" of the region under inspection. This is usually done by using a combination of various approaches. The tracker must be accurate and fast for suitable implementation.

After evaluating a set of different feature trackers, Kernalized Correlation Feature(KCF) tracker was chosen. The KCF is a variant of correlation filter. In a correlation filter, correlation between two samples is taken and when the samples match, the correlation is highest. This method is also applied in the process of tracking. Upon providing a sample patch, the filter can look for a patch in a neighbourhood for a patch having the highest correlation between the two. This is done over various frames in time so as to get a continuous track of the object under supervision. In a standard tracker, performance degrades substantially if the appearance of the object changes significantly in subsequent frames.

In KCF, however, this is not the case. KCF consists of a linear regression model which keeps updating the appearance of the object. This property of the KCF tracker makes it robust. The linear regression model being implemented is also very lightweight i.e. is implemented in the form of circular matrices and kernel (often Gaussian) functions.This makes the process much more lightweight and thus does not take much processing power to compute. This is an advantage in the trade off between accuracy and computational power required. The entire process is depicted in Fig 2.5.

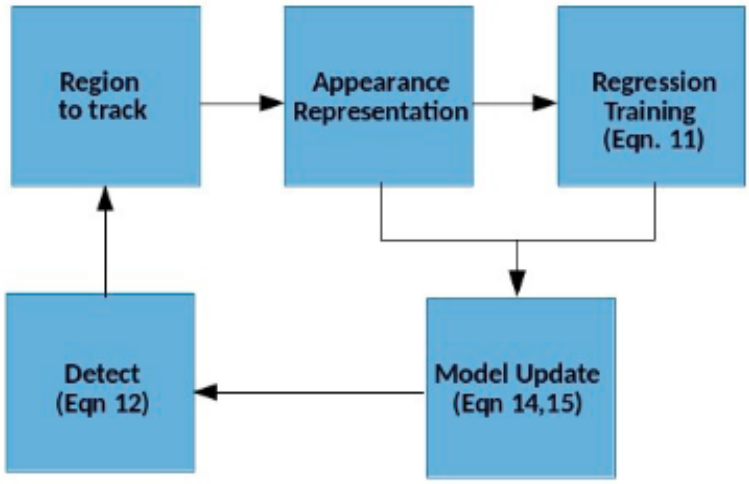

Fig. 2.5 The Block Diagram of KCF Tracker

2.2.4 Histogram of Oriented Gradients(HOG) + Support Vector Machine

HOG is a feature descriptor. A feature descriptor extracts and summarises the information held in an image in the form of a conclusive form. This makes the processing much more efficient as processing all pixel values is not necessary. An HOG descriptor extracts and summarises information about the image gradients along the axis of the image. The gradients are useful for detecting rapid changes in the image in terms of edges, corners etc. The descriptor then summarises these gradients in the form of 8 bins (as decided by the author of this algorithm). The edges and corners are preferred because they contain a lot more useful information than plain surfaces with little to no variation.

An SVM is a machine learning algorithm which is used in classification tasks. In the hyper-plane of the data points, a regression algorithm is used to determine a boundary (support-vector) which most accurately segregates these points into different classes. In the current use case, the SVM would be trained on HOG features. This would be thereafter be labelled into 2 classes based on if there exists a car or not based on these HOG features.

To inculcate localization of the vehicle, the entire system (HOG+SVM) is run over multiple smaller segments in a specific region of interest. This is also done over various scaling sizes in order to take into consideration the changing sizes of vehicles depending on the distance from the camera.

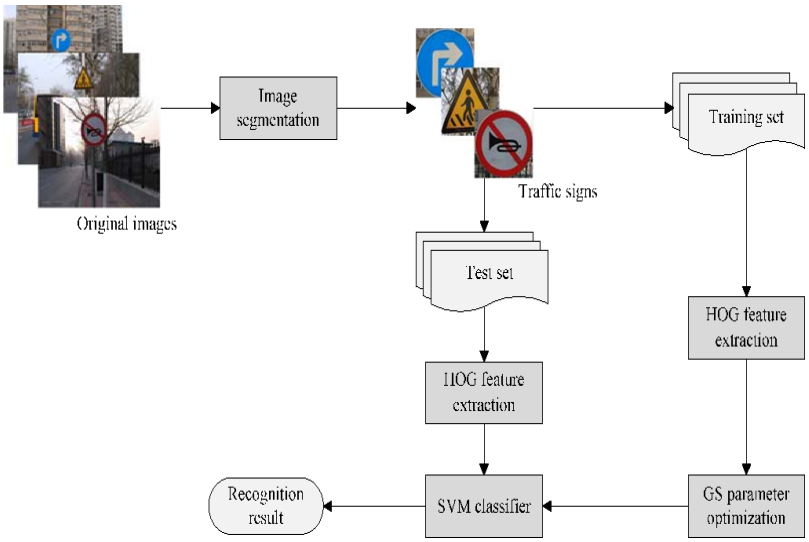

Fig. 2.6 The Block Diagram of HOG+SVM Pipeline

3. *Implementation*

After evaluation of the aforementioned algorithms and evaluating the applicability in the given conditions, it was decided that **Haar Cascades** coupled with other algorithms, namely **Optical Vector Flow** and **Kernelized Correlation Filter Tracker** would be the most appropriate and applicable to this use case.

3.1 Final Approach

The final approach which was implemented was a combination of using Haar Cascades and the KCF tracker in conjugation. This was done to make the tracking done by the Haar Cascade more robust to varying sizes and perspectives of cars.

Since the Haar Cascade was trained on a dataset which consisted of vehicles at an appreciable distance from the camera, if a vehicle came too close to the camera, the Haar Cascade failed. In order to overcome this limitation of the Haar Cascade, it was coupled with a KCF tracker. This tracker would take in the localised objects from the Haar Cascade and then would track them till the time the tracker was reset.

In order to track various vehicles at the same time, a multi-object tracker was used. To this tracker, the vehicle's co-ordinates were added and thereafter tracked. To further improve this approach and process, suggestions are given in **Section 5 - Future Development**.

3.2 Training the Haar Cascade

To train the Haar Casacade model, OpenCV provides in built functionality. These functionalities include functions like *opencv_createsamples, opencv_annotation, opencv_traincascade* and *opencv_visualisation.*

As metioned before, to train a Haar Casacde, a set of *positive samples* and a set of *negative samples* are required. There is a possibility that the number of images in the trainging data do not suffice. For this situation, it is suggested to use the *opencv_createsamples* functionality.

This augments the images to create more samples from the already existing training data samples. For the positive images, a bounding box's coordinates are required to be given. This bounding box would contain regions where the object of interest (in this case, cars) occur in that image. This information must be stored in a *.dat* file which will be used later during training. For the negative samples, no such thing is needed to be done. This process of marking can also be done by the *opencv_annotations* tool which has a GUI interface. Using this is suggested as it makes the process faster and more accurate.

Once all the training data has been prepared, training of the *haar_cascade* begins. the *opencv_traincascade* functionality is used for this purpose. After providing the relevant command line arguments like *data, bg, vec* amongst a few, a *Haar Cascade* can be successfully trained. For complete in depth documentation of all these functionalities, it is recommended to refer to the OpenCV documentation (as given in **Section 7 - References**).

3.3 Implementation

The process as described in the above defined algorithm is demonstrated in the code below

---

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <opencv2/tracking.hpp>
#include <opencv2/core/ocl.hpp>
#include <opencv2/core/utility.hpp>
#include <opencv2/videoio.hpp>
#include <cstring>
#include <ctime>
#include "samples_utility.hpp"

#include<iostream>

#define RESET   "\033[0m"
#define RED     "\033[1;31m"      /* Red */
#define GREEN   "\033[32m"      /* Green */
#define YELLOW "\033[33m"  /* Yellow */

using namespace std;
```

```
using namespace cv;

CascadeClassifier car_cascade;


int main( int argc, const char **argv)
{
    const String keys =
    "{help h usage ? |   | Select Source File(-s) and
                          Cascade file (-c) }"
    "{c              |cars4.xml | Cascade }"
    "{s              |file7.mp4 | source Input }"
    ;


    CommandLineParser parser(argc, argv, keys);

    parser.about("ADAS-v1.0.0");

    if (parser.has("help"))
    {
        parser.printMessage();
        return 0;
    }

    String car_cascade_name = parser.get<String>("c");

    if (! car_cascade.load(car_cascade_name))
    {
        cout <<"[ERROR] Cascade Not Found"<<endl;
    }

    VideoCapture capture; //declare a video capture object

    String camera_device = parser.get<String>("s");
    cout <<camera_device <<endl;
```

```
//For using camera
//VideoCapture capture("imxv4l2videosrc  device=/dev/video0 ! queue !
videoconvert ! appsink emit-signals=true drop=true max-buffers=1
sync = false",CAP_GSTREAMER);

//for using file
//VideoCapture capture("file7.mp4");

//for using network streamed video
VideoCapture capture("udpsrc port=5200 !  application/x-rtp,
encoding-name=JPEG,payload=26 ! rtpjpegdepay ! jpegdec !
videoconvert ! appsink sync=false",CAP_GSTREAMER);


if (!capture.isOpened())
{
    cout <<"[ERROR] Video Not Found"<<endl;
}

Mat frame;
Mat gray;
Mat cropped;
int w, h, x, y;
bool flag;

//create tracker pointer


String trackingAlg = "KCF";
MultiTracker trackers(trackingAlg);

cout<<RED "Initialising Complete" RESET<<endl;

int frameCount = 0;
```

```
while( capture.read(frame))
{

    //Define ROI
    int rows = frame.rows;
    int cols = frame.cols;

    Point points[1][5];
    points[0][0] = Point( 0, rows);  //bottom left (x,y)
    points[0][1] = Point(0, 400); //top left (x,y)
    points[0][2] = Point( cols/8+100, 400); //top right (x,y)
    points[0][3] = Point(cols/8+100, 550);
    points[0][4] = Point(cols/8+400,rows); //bottom right (x,y)

    const Point* ppt[1] = {points[0]};
    int npt=5;


    if(frame.empty())
    {
        cout <<"[ERROR] No Frame Captured"<<endl;
    }

    //Instantiate every 40 frames, a multi object tracker
    //which will track all objects as given by the Haar Classifier
    if (frameCount%40 == 0)
    {
        MultiTracker trackers(trackingAlg);
    }

    //Check for cars every 10 frames
    if(frameCount%10 == 0)
    {
        Mat gray;
        Mat cropped;
        Mat mask = Mat::zeros(cvSize(cols, rows),CV_8UC1);
```

```
        cvtColor(frame, gray, COLOR_BGR2GRAY);
        equalizeHist(gray, gray);

        fillPoly( mask,ppt, &npt,1,Scalar( 255, 255, 255 ));
        bitwise_and(gray, mask, cropped);

        imshow("Cropped", cropped);

        std::vector<Rect> cars;

        objects.clear(); //let go of old tracks
        algorithms.clear();

        car_cascade.detectMultiScale(cropped, cars,
                                     1.4,
                                     3,
                                     0,
                                     cvSize(50,50));
        for (size_t i = 0 ; i < cars.size(); i ++)
        {
            Point center(cars[i].x+cars[i].width/2,
            cars[i].y+cars[i].height/2);
            Point vertex1(cars[i].x, cars[i].y);
            Point vertex2(cars[i].x+cars[i].width,
            cars[i].y+cars[i].height);

            //show centroid and the bounding box of the car
            circle(frame, center, 1, Scalar(255,0,0), 2);
            rectangle(frame, vertex1, vertex2, Scalar(0,0,255),2);

            //create a bounding box and add it to the tracker
            trackers.add(frame, cars[i]);
        }
    }
```

```
    for(unsigned i=0;i<trackers->getObjects().size();i++)
        {
            //rectangle( frame, trackers->getObjects()[i],
            //Scalar( 255, 255, 255 ), .5, 1 );
            //bbox[i] = trackers->getObjects()[i].x;
            w = trackers->getObjects()[i].width;
            h = trackers->getObjects()[i].height;
            x = trackers->getObjects()[i].x;
            y = trackers->getObjects()[i].y;

            //Algorithm for warning the driver
            if ( (w  < 90 && w > 60) || (y + h < rows-50 &&
                y+h > rows - 200))
            {
                cout <<YELLOW "[WARNING] VEHICLE APPROACHING" RESET<<endl;
                rectangle(frame, cvPoint(0,0), cvPoint(cols, rows),
                Scalar( 0, 255, 255 ), 3, 1.5 );
            }
            if (w > 90 || (y+h > rows-50))
            {
                cout <<RED "[ALERT!] STOP!" RESET<<endl;
                rectangle(frame, cvPoint(0,0), cvPoint(cols, rows),
                Scalar( 0, 0, 255 ), 3, 3 );
            }
        }

    imshow("Detected", frame);

    frameCount++;

    if(waitKey(1) == 27)
    {
        break;
    }
}
```

```
    return 0;
}
```

4. *Implementing it on the Board*

4.1 About the Board

- **Processor**: i.MX 6Quad 1 GHz ARM®Cortex-TM-A9 processor
- **Memory**: 2 GB DDR3 SDRAM up to 533 MHz (1066 MTPS) memory,8 GB eMMC Flash, 4 MB SPI NOR Flash, 256GB Memory
- **Hardware Interface**:mPCIe connector, 2x full-size SD/MMC micro card slots, 7-pin SATA data connector (i.MX 6Quad only), 10/100/1000 Ethernet port, 1x USB 2.0 OTG port (micro USB), 1x USB 2.0 debug port (micro USB), 1x serial-to-USB, 4x FAKRA connectors for connecting cam and 3x HSD connecterasors for interfacing screen displays. (for JTAG) HSD
- **Display**: 10.1" 1024 x 768 LVDS display with integrated P-cap sensing HDMI connector, LVDS connector (for optional second display), LCD expansion connector (parallel, 24-bit), EPDC expansion connector (i.MX 6DualLite only), MIPI DSI connector (two data lanes, 1 GHz each)

4.2 Implementation

The algorithm was written in C++ with OpenCV-3.4.4. The board had OpenCV-4.0.0 pre. To port the algorithm, a few changes had to be made. After making these changes, the algorithm was implemented again but a few problems in terms of bottlenecks were experienced. For proof-of-concept, the algorithm was demonstrated using prerecorded videos instead of live streams.

Subsequently the algorithm was also tried to implement with the existing architecture of the other project being developed by Karel Electronics.

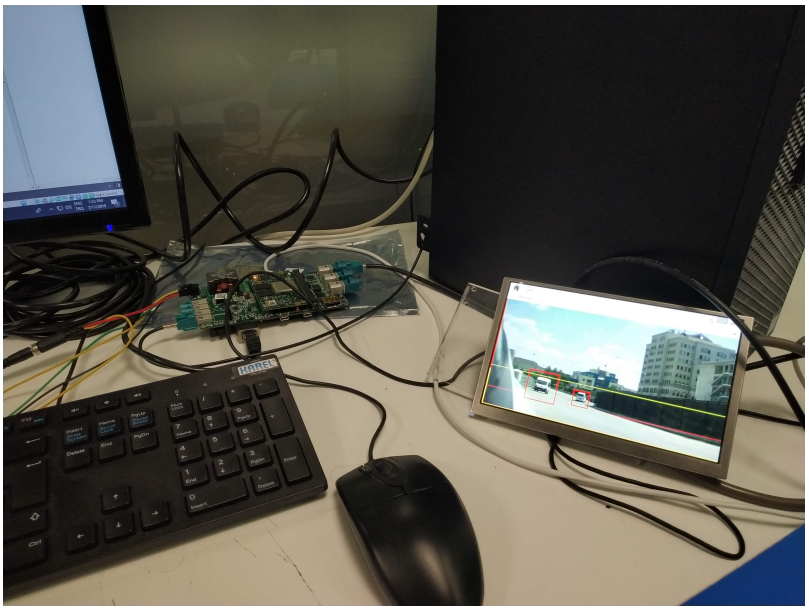

Fig. 4.1 Embedded Board

4.3 Bottlenecks and Workarounds

4.3.1 Pipeline Bottleneck

To make the project compatible with the preexisting pipeline architecture of the project, modifications were made to the code to integrate these features. This was done by using the parameters being provided to the *VideoCapture* function of OpenCV. However, in order to integrate the pipeline structure of GStreamer, OpenCV only provides support only for GStreamer's *appsink* only. This was found to be slow and had a lag of about 0.8 seconds from ground truth to screen.

To overcome this lag, the car detection algorithm had to be integrated into the existing pipeline which had the board's custom sink (*imxipuvideosink*). This was a challenge as by default this sink does not allow information sharing to other parallel programs. For this issue, after consulting and getting help from more experienced members of the team, a workaround was worked out. This workaround was to extract images from the frame buffer of the sink and perform the algorithmic operations followed by writing it to the disk. This implementation would result to an output at the output terminal warning the driver.

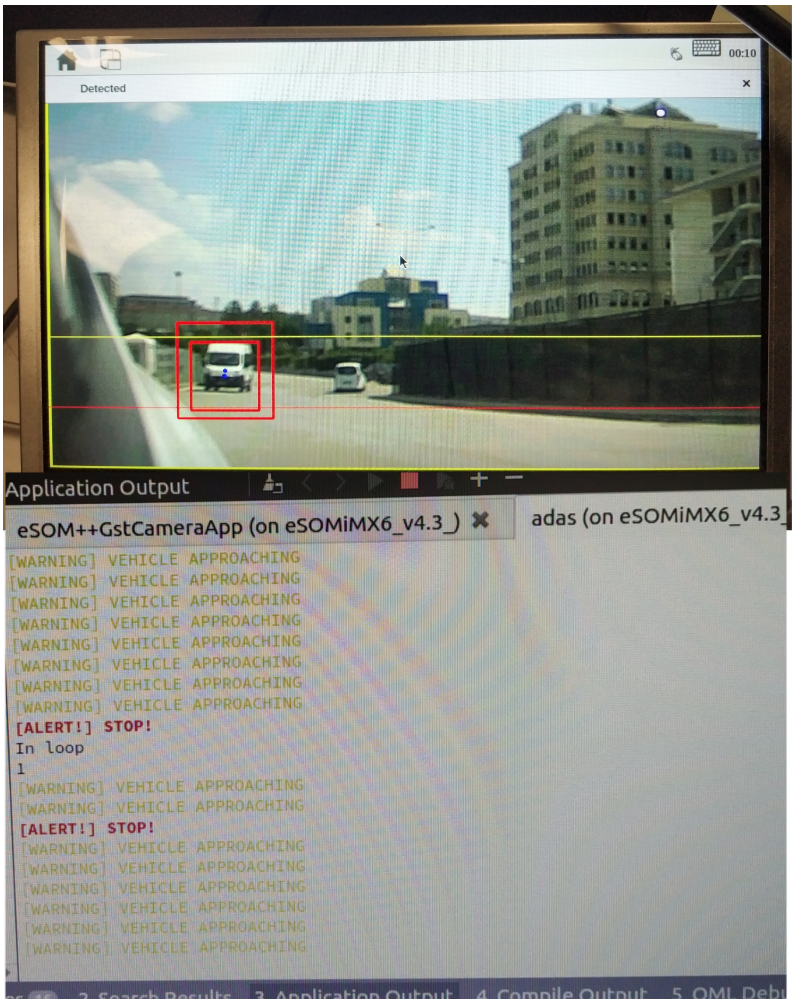

Fig. 4.2 Output Obtained on Board

4.3.2 Processing Power

The algorithm was tried again with both cameras being interfaced at separate ports. This now posed another problem which was processing power.

To overcome this problem, the code was modified so as to look for cars in a specified region of interest (ROI). Also, the process of detection and tracking was done every few frames instead of doing it every frame. Cars were tracked every 10 frames and the multi object tracker was refreshed (from memory) every 40 frames in order to keep the process updated about the new cars as time progresses.

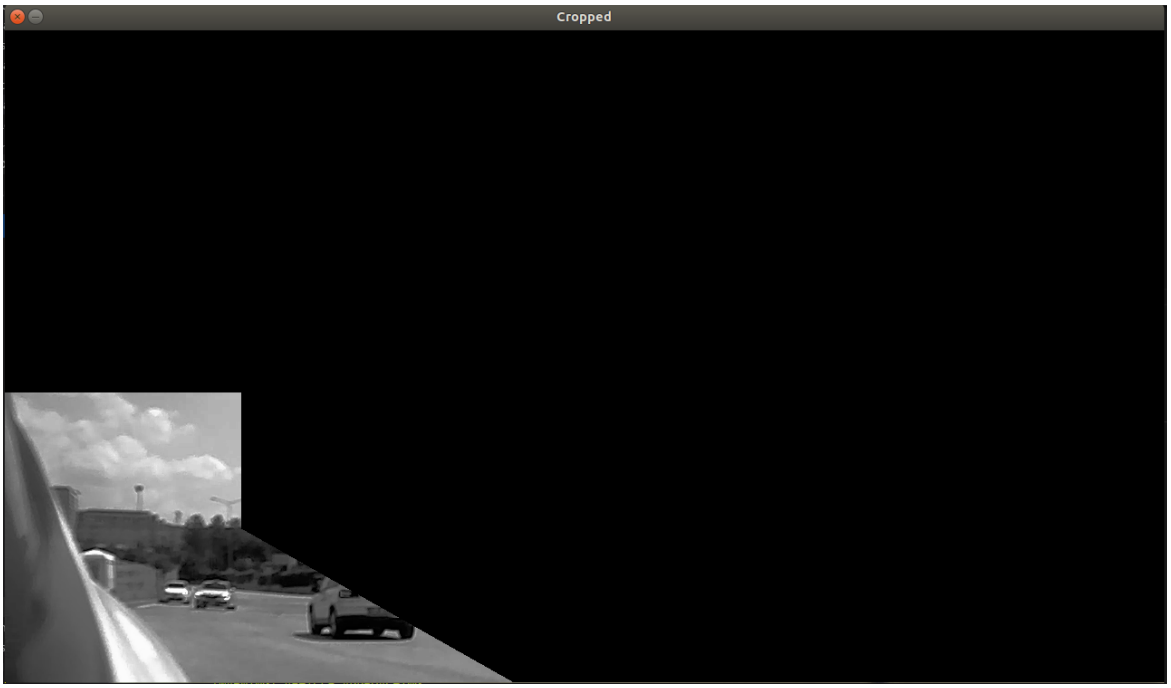

Fig. 4.3 Cropped ROI

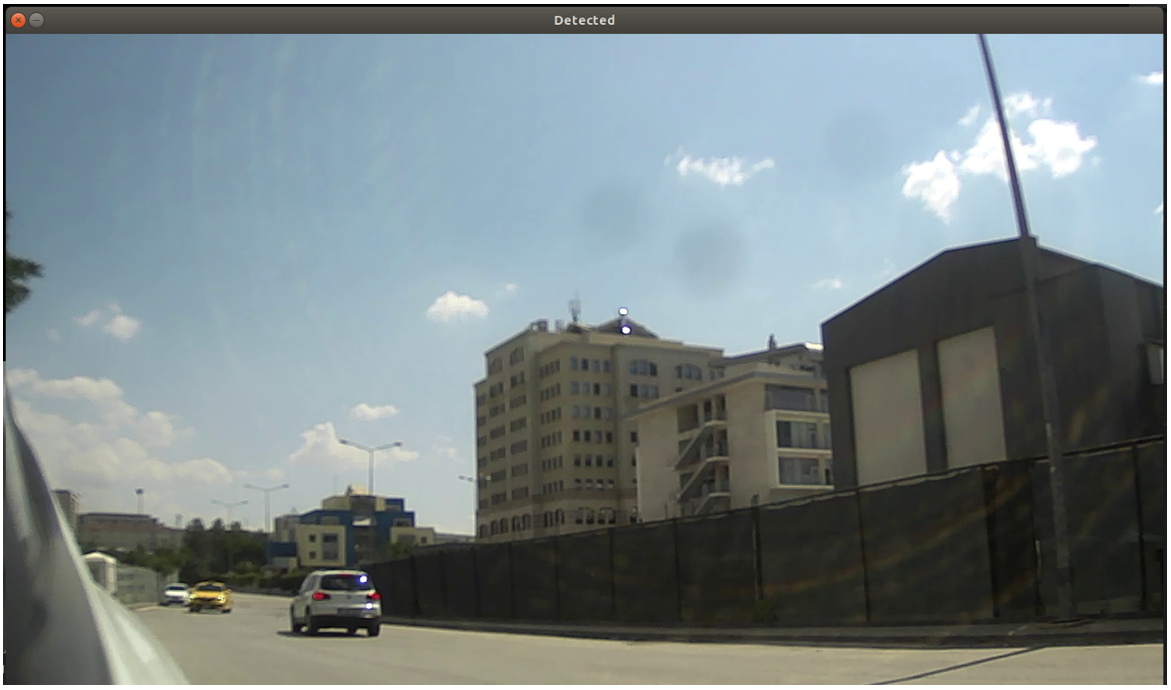

Fig. 4.4 Complete Frame

4.3.3 Displaying on Frame Buffer 1

The alert post detection was aimed at being displayed on frame buffer 1 while the video stream played on frame buffer 0. This was not implemented completely successfully due

to problems with integrating the algorithm with the pipeline. The official documentation of GStreamer did not address all the issues faced during implementation. However, frame buffer 1 was available to show video stream through pipeline directly.

To overcome this problem, a *tee* structure was initialised and the processed frames were pushed onto the pipeline in order to show them on frame buffer 1. However, this lead to errors during run-time which could not be diagnosed even after brainstorming with the team.

5. *Results and Future Development*

5.1 Results

Using Haar Cascades and KCF Trackers, an accurate model was developed. This model was robust to lighting conditions, changing scenarios and rapid changes in environment. The results were displayed on the output terminal and as an outer boundary on the videos. Some of the outputs obtained are shown as below

Fig. 5.1 Borders turn Red to Indicate a Vehicle in Close Proximity

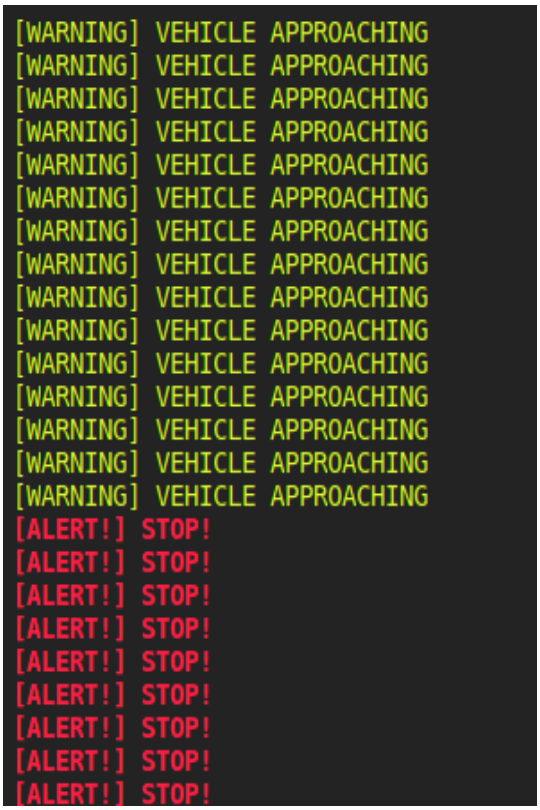

Fig. 5.2 Output Obtained on the Output Console (Terminal)

5.2 Training Haar Cascades on Custom Data

Currently the Haar Cascade being used has been trained on public datasets which do not scale well to the current application. In order to improve the performance of the system, a custom dataset can be collected by the camera while being mounted on the car. This would lead to a more relevant dataset that suits the purpose better. This would also mean better accuracy and possibly eliminating the KCF Tracker since the Haar Cascade can itself keep a track of incoming cars. This would in turn lead to better efficiency.

5.3 Implementing Pedestrian Detection

The current code supports only detection of cars. The code can also be modified to inculcate the detection of humans to help the driver be better aware of their surrounding. This can be done via either using Haar Cascades or using Histogram of Gradients (HoG) as a feature extractor for a Support Vector Machine (SVM). This SVM can thereafter warn the driver about any pedestrian that is at a risk from the vehicle.

5.4 Driver Awareness System

A conjugate system can be developed for the driver himself rather than the surroundings. It can be developed to track if the driver is focusing on the road or is distracted in some way i.e. not paying attention on the road, sleeping while driving etc. This can be done by extracting facial features (using dlib library) and thereafter tracking the movement of the face and eyes to check if the driver is aware of the system. If the driver is not aware, there can be a beeping alarm for the driver to pay attention or the car can eventually slow down and stop while indicating to the traffic to continue from around.

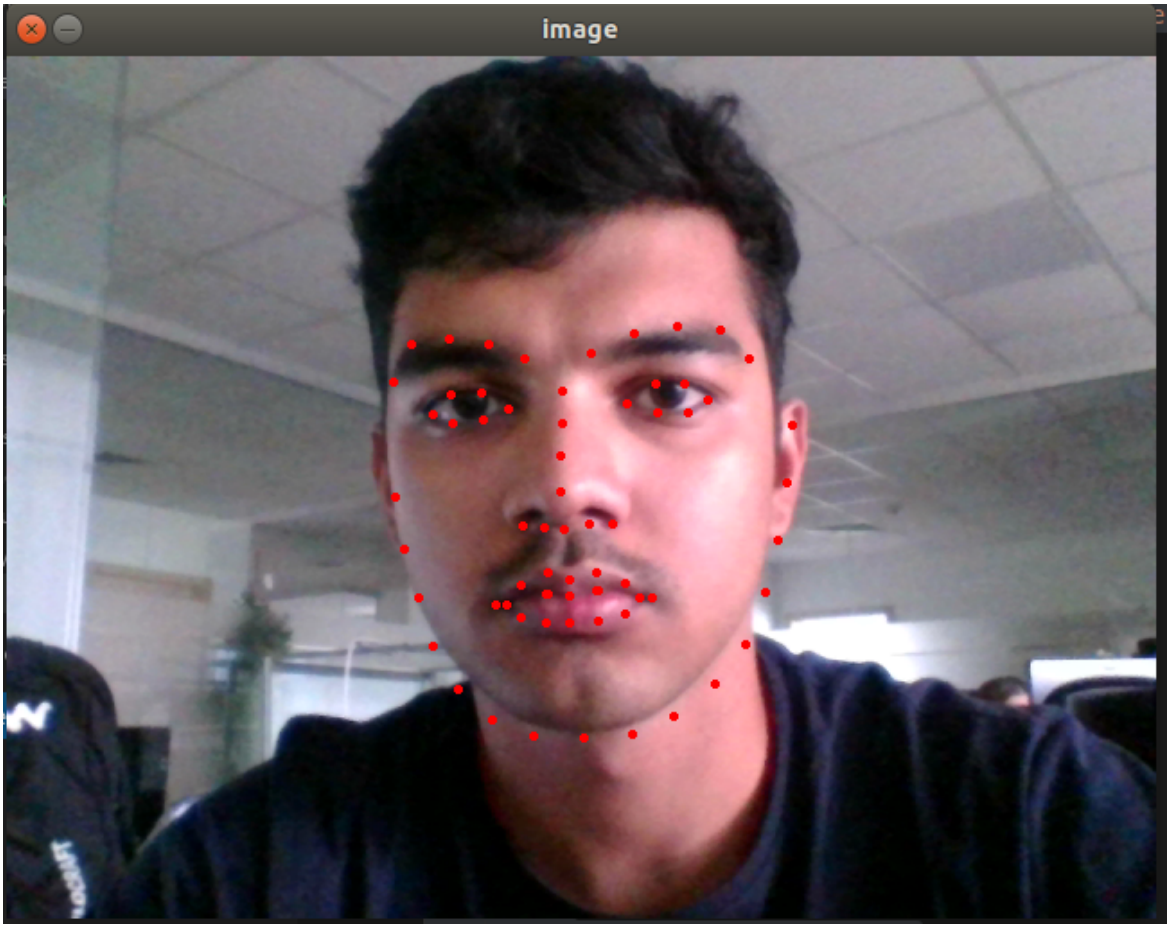

Fig. 5.3 Facial Tracking

6. *Conclusion*

Over this period of 15 days, several algorithms were examined and the most appropriate were chosen and implemented. Barriers experienced were overcome with the guidance and skill set of experienced professionals in the form of Company employees.

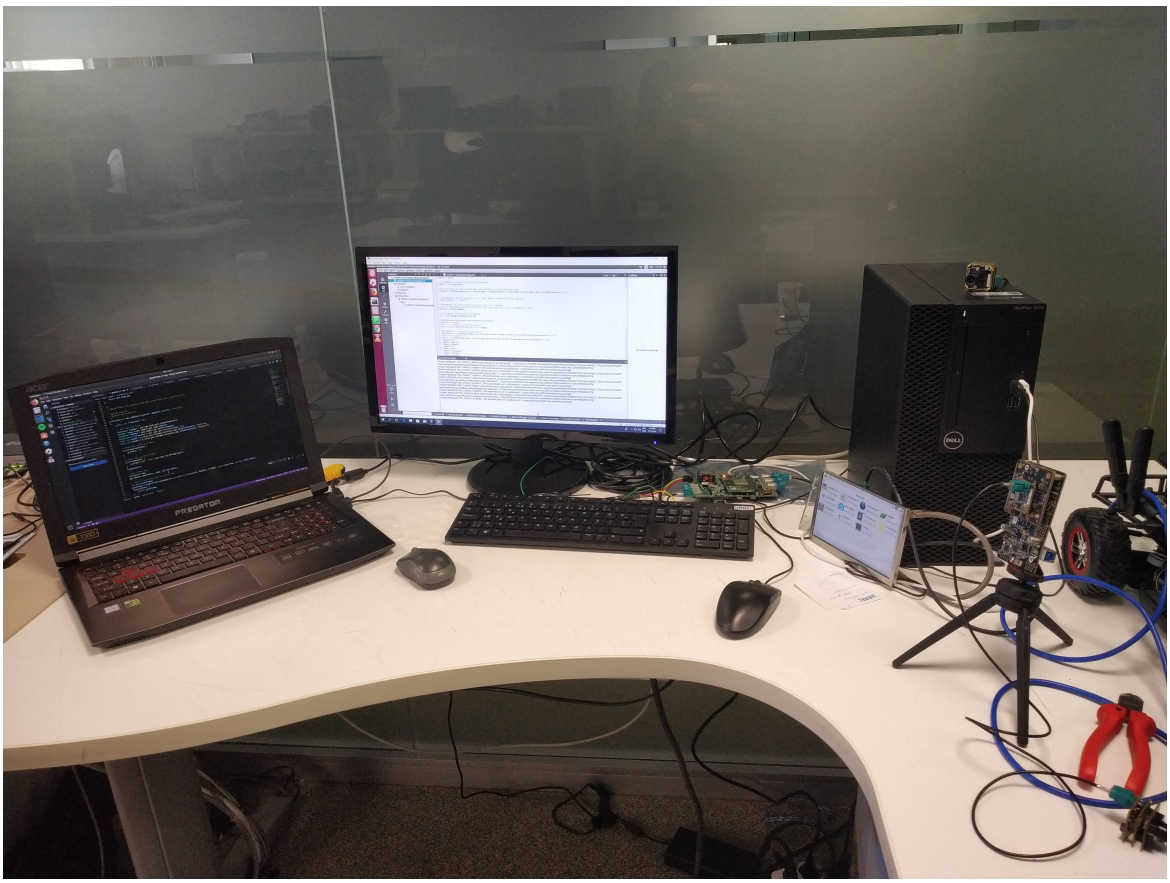

Fig. 6.1 My Workstation

The overall experience at Karel has been a very informative and memorable one. The technical work coupled with the friendly and warm behavior of the entire team has been one of the best environments I have worked in. I have learnt so much more than just the

technicalities that were involved with the project. The team members have always been welcoming towards any questions that I might have had, They always made time for my queries despite their schedules. This welcome environment has only helped me grow in ways that were unknown to me till now and I will always be grateful for this experience.

7. *References*

1. **Rapid Objec tDetection using a Boosted Cascade of Simple Features** by Paul Viola, Michael Jones https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

2. **Optical FlowMeasurement using Lucas kanade Method** by Dhara Patel,Saurabh Upadhyay https://pdfs.semanticscholar.org/6841/d3368d4dfb52548cd0ed5fef29199d14c014.pdf

3. **KCF Tracker** by João F. Henriques, Rui Caseiro, Pedro Martins, Jorge Batista http://www.robots.ox.ac.uk/~joao/circulant/

4. **KCF Filter Performance Evaluation** by Miacheal George, Babita Jose, Jismon Mathew https://www.sciencedirect.com/science/article/pii/S1877050917328363

5. **HoG + SVM for Object Detection** https://medium.com/@mithi/vehicles-tracking-with-hog-and-linea

6. **Optical Flow Explanation** by Ce Liu, Steve Seitz, Larry Zitnick and Ali Farhadi https://homes.cs.washington.edu/~shapiro/EE596/notes/Optical_Flow.pdf

7. **How to Train a Haar Cascade** https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html