Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Final Report 2017

Project Title:	**Augmented Reality for Driving Applications**
Student:	**Alessandro Versini**
CID:	**00826818**
Course:	**EEE4**
Project Supervisor:	**Dr Yiannis Demiris**

Abstract

The project concerned the development of an application for a pair of AR glasses that could augment the reality of the driver without requiring special hardware installation on the car. Such requirements are defined in order to make the AR experience portable from vehicle to vehicle and significantly cheaper in terms of development costs.

The difference between our work and that previously published in the field of Computer Vision (CV) applied to driving is in that we do not make any assumptions on the pose of the camera, which is incorporated in the pair of glasses and hence is free to move inside the car cockpit. To appreciate the implications of a moving camera inside the cockpit we built an ad hoc dataset which we publish alongside this report.

More specifically, our literature review illustrated that a large number of existing CV methods for the detection of road objects make strong assumptions on the mounting of the camera. Therefore, the first part of our work aimed at bridging the gaps between such existing CV algorithms and our novel moving camera setting by tracking and compensating for the driver's head movements. In the second part, we proved that once the constantly varying orientation of the camera is taken into account and corrected, standard state of the art techniques can be used as a ground for building AR applications.

As a result, the software that we produced during our research and development is able to augment the reality of the driver using only an unconstrained mono-camera and a IMU. It is capable of detecting any type of road lane and surrounding vehicles and is able to generate warnings (e.g. lane departure, vehicle collision, overspeed, etc.) in a way that is informative but not distractive for the driver. The software has been tested on data collected from real driving sessions.

Acknowledgements

I would like to thank my academic supervisor Prof. Yiannis Demiris for accepting to look after an industrial project and for supporting me throughout the several months of research and development.

Furthermore, I would like to acknowledge Jörg Preissinger from BMW Augmented Reality Research for giving me the opportunity to work on a project of interest to BMW Group Research.

Finally, a special thank goes to my tutor Prof. Peter Cheung, to the director of undergraduate studies Dr. Kristel Fobelets and to the whole Imperial College EE staff for conveying me such a deep passion for engineering.

To my family and beloved ones for supporting me throughout these exciting and intense four years at Imperial College London.

Contents

I Project Core **3**

1 Introduction & Requirements **4**
1.1 Introduction 4
1.2 Requirements 4

2 Literature Review **5**
2.1 Road Features Detection 5
2.1.1 Vision-based Lane Detection 5
2.1.2 Object Detection 10
2.2 Head Tracking 12
2.2.1 Attitude 13
2.2.2 Position 14
2.2.3 Pose 15
2.3 Review of Commercial AR Devices 15
2.3.1 Microsoft HoloLens 15
2.3.2 ODG R7 Smart Glasses 16
2.3.3 Comments 16
2.4 Comments 17

3 System Design **18**
3.1 Introduction 18
3.2 Data Acquisition 19
3.2.1 Data Set 20
3.3 Pose Determination 20
3.3.1 Naive Head Tracking 23
3.3.2 Robust Head Tracking 32
3.3.3 Comments 40
3.4 ROI Extrapolation 41
3.5 Road Features Extraction 43
3.5.1 Lane Detection 43
3.5.2 Multi-Class Object Classifier 53
3.6 User Interface 59

4 System Implementation & Demonstration **62**
4.1 Python Implementation . . . 62
4.1.1 Libraries . . . 62
4.1.2 Components . . . 62
4.2 Demo . . . 66
4.3 Code Profiling . . . 68
4.3.1 Head Tracking . . . 68
4.3.2 Lane Detection . . . 69
4.3.3 Multi-Class Object Classifier . . . 69
4.3.4 UI . . . 70
4.3.5 Summary . . . 70

5 Comments & Future Work **71**
5.1 Comments . . . 71
5.2 Further Development Opportunities . . . 72
5.2.1 Pose Determination Using PnP . . . 72
5.2.2 Innovative User Interface . . . 74
5.3 Applications . . . 74
5.3.1 Research . . . 74
5.3.2 Driving School . . . 74
5.3.3 Marketing . . . 74

II Extras **75**

6 Data Collection **76**
6.1 Introduction . . . 76
6.2 iPhone Application . . . 76
6.2.1 Camera . . . 77
6.2.2 Position & Motion Sensors . . . 78
6.2.3 Log-Files Management . . . 79
6.2.4 Example . . . 79
6.3 Camera Calibration . . . 80

A Code **85**

B Dataset **86**

C Project Proposal **87**

Part I

Project Core

Chapter 1

Introduction & Requirements

1.1 Introduction

This project has been designed in collaboration with Jörg Preissinger from BMW Augmented Reality Research and is supervised by Prof. Yiannis Demiris at Imperial College London. This document provides a final report of the research and development conducted between October 2016 and June 2017.

1.2 Requirements

The aim of the project is to design a system that could be used to enhance the driving experience in real time through Augmented Reality (AR) technology. The challenge that is proposed by BMW is to design software for a stand-alone head mounted device that does not rely on any existing car's on-board system. Installation of additional hardware on the vehicle is also against requirements.

The use of augmented reality in commercial cars has been a widely researched topic, especially in order to support the development of head up displays for vehicle's cockpits. The novel application of such technology in the form of head mounted devices will be able to extend the AR experience beyond the driver (e.g. to the other passengers) and even beyond the car cockpit itself (e.g. first and last mile pedestrian navigation).

Clearly, AR in the form of head mounted devices also generates some new complications such as the necessity to track the position and orientation of the head inside the cockpit in order to achieve the same level of functionality as a cockpit mounted head up display. We will discuss how we approached this one and many more design issues throughout this report.

Chapter 2

Literature Review

This project spans a wide range of research topics. In this report we will be mainly concerned with computer vision applied to road feature detection and tracking, camera pose estimation through vision and inertial measurements and finally techniques to display AR content to the user of our system. Clearly, in order to tackle a project of such wide scope, we broke it down into smaller areas of interest which we are going to cover one at a time before combining them together into our final design.

2.1 Road Features Detection

With the term road features we refer to objects of interest that are typical of road scenes such as traffic lanes, cars, pedestrians, traffic signs, etc. In particular, during our research, we started with the problem of traffic lane detection and tracking and only subsequently we moved our focus towards the identification of surrounding objects such as pedestrians and cars.

It seemed logical to proceed in this order since, in an analogy with classical Computer Vision in a closed environment, this would correspond to the detection of the frame (e.g. room walls, ceiling, floor, etc.) first, followed by the identification of the objects that populate the environment (e.g. chairs, people, etc.).

2.1.1 Vision-based Lane Detection

Definition of Requirements

Vision-based road lane detection has been a very popular subject of research in the last couple of decades and several papers have been proposed to document findings over these years. However a common assumption made by the literature is that the camera is installed steadily on the car in a carefully measured position [15][32][1]. This type of solution, despite making the image processing easier and

more reliable, is clearly not applicable to an head mounted set which is expected to move in all its 6 degrees of freedom.

It follows that it would be ideal if there existed a lane detector that was tolerant to relaxed assumptions on the position of the camera. Furthermore it would also be desirable for it to work reliably on different types of road lanes (e.g. curved, dashed, etc.).

Review of Existing Methodologies

According to Kumar and Simon [15], among vision based lane detection methods, two types of algorithms exist:

- feature based algorithms which use low level features (e.g. lines, edges)
- model based algorithms which combine a-priori mathematical models with observations from the camera

However, in practice, most of the solutions presented in popular papers seem to combine elements of both worlds.

Besides the classification of algorithms, Kumar and Simon also provide a brief analysis of the most common challenges encountered in detecting road lanes. In particular they quote four which are illustrated with the aid of Figure 2.1 (from their paper). The authors claim that in ideal conditions the task of detecting road lanes is a simple one; however, the presence of adverse conditions (e.g. weather, occlusion by obstacle, shadows, etc.) can very quickly degrade the algorithms' performance.

(a) Vehicle occlusion

(b) Shadows

(c) Weather conditions

(d) Uneven lighting

Figure 2.1: Four examples of challenges in lane detection as suggested by Kumar & Simon [15]

Consistently with our definition of requirements, our initial research into lane detectors looked into methods that did not heavily rely on a particular mounting of the camera. Nevertheless, unfortunately, among existing lane detection methods, very little could be found that met the desired requirements.

In particular, it is very common in literature to assume that a Region Of Interest (ROI) can be statically extracted from the input frame. If appropriately chosen, such ROI can make the lane detection much more effective as well as computationally efficient. Most of the time, the ROI is further divided into a lower ROI where lane boundaries are approximated as straight and an upper ROI where curved lane boundaries are searched for. An illustration is shown in Figure 2.2

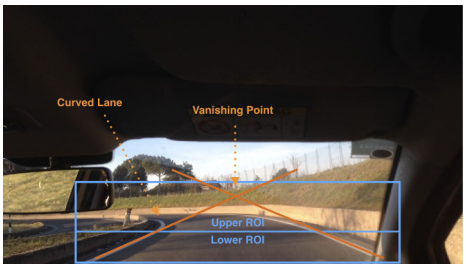
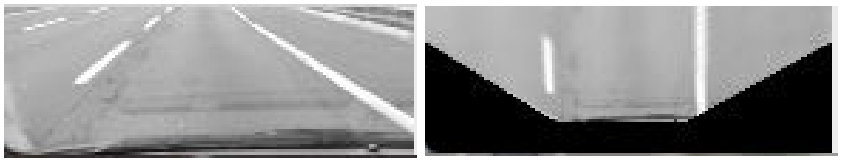


Figure 2.2: Example of statically extracted ROI from a steadily mounted camera frame.

Despite the common framework, every paper in the literature then tackles the problem in its own way. Generalising, we could identify five stages in the lane detection which have been solved using a number of different techniques.

- (Optionally) ROI transformation
- Filtering
- Straight lane candidate points
- (Optionally) curved lane candidate points
- Interpolation

ROI Transformation In this first stage the region of interest undergoes an inverse projective transformation as shown in Figure 2.3. Such transformation is only occasionally used and it makes very strict assumptions on the mounting of the camera. It has the advantage of improving the computational efficiency of the lane detection algorithm [1].



(a) Original Frame (b) Inverse Perspective Transformation

Figure 2.3: Example of ROI transformation preceding lane detection.

Filtering In the filtering stage, lanes are isolated from the darker[1] road background. Methods proposed include gray-scale thresholding, steerable filters, edge detection, etc. Figure 2.4 shows an example of dynamic threshold and

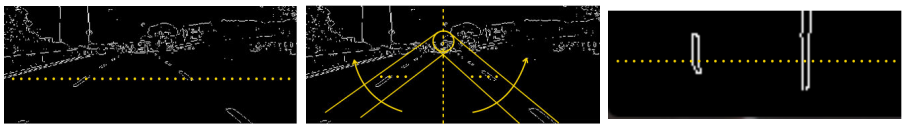
[1]It can be assumed that lanes are always of a lighter colour than the tarmac in the background although not necessarily white.

Canny edge detection applied to an extracted region of interest. The dynamic threshold is chosen to achieve maximum separation between the lane and background pixel intensities.

(a) Canny Edge (b) Dynamic Threshold (c) Dynamic Threshold

Figure 2.4: Examples of filtering preceding lane detection.

Straight Lanes Candidate lane boundary points for straight lanes are usually searched for in the lower subregion of the ROI. Again, several methods have been proposed. Such include horizontal pixel scans, Hough transforms, vanishing point based detection, etc [9][12]. It is usually the case that the more computationally efficient methods are those that make the more strict assumptions on the camera mounting. Figure 2.5 shows a few examples of straight lane detection methods.



(a) Horizontal Pixel Scan (b) Vanishing Point (c) Horizontal pixel scan

Figure 2.5: Illustration of straight lane detection methods.

Curved Lanes Curved lanes are not always supported by lane detectors. For example, in lane detectors designed for lane departure systems, the shape of the road further away from the camera is not of particular interest. Methods to detect curved lanes include horizontal pixel scans, calculation of slopes of hough lines, river-flow methods, etc [7].

Interpolation Interpolation is the last step of the lane detection and allows to calculate an equation of the two lane boundaries in the image coordinate system. Interpolating curves include generic polynomials, conics (hyperbolae or circles under inverse projective transformations), cubic spilines, etc. [32][35].

Comments

Our research showed that it is indeed quite difficult to meet the desired requirement of a lane detector which is independent on the mounting of the camera. There are methods which are likely to be more or less sensitive to it but all of them rely on this assumption in some way [32][35][9][12].

These findings encouraged us to look at ways of compensating for the movements of the camera by tracking the position and orientation of the driver's head inside the cockpit.

2.1.2 Object Detection

To simplify the problem slightly, we have concentrated our efforts on a car's most frequent sources of interaction: other vehicles and pedestrians. Hence, in what follows we present a brief review of the state of the art vehicle and pedestrian detectors.

Vehicle Detection

In their review of on-board vehicle detection systems, Sun et al. [31] observe that most existing designs divide the task into two complementary steps. In the first, Hypothesis Generation (HG), candidate vehicles are found based on high level hypothesis on their shape, colour, motion, etc.; in the second stage, Hypothesis Verification (HV), the candidates found in HG are then verified by means of more accurate and computationally expensive methods.

Because Hypothesis Generation methods yield results which are then validated in the following stage, they are designed to be fast and tend to generate an high rate of false positives. As mentioned, the speed of HG is achieved through the choice of low complexity (although inaccurate) algorithms to identify vehicles' features in a test image. More specifically, the authors divide the methods for HG into three classes: knowledge-based, stereo-based, motion-based.

Knowledge-based method use bi-dimensional information such as 2D shape (i.e. corners, edges), colour, texture and even vehicle lights (in low ambient light condition, e.g. night). Stereo-based and motion-based methods on the other hand use 3D information to determine the presence (and position) of a car in an image. Stereo-based techniques, as the name suggests, use a double camera setting and calculate the depth of several planes in the image; motion-based methods extrapolate information about the depth through techniques such as the optical flow [10].

Similarly to HG methods, HV methods can also be divided in multiple classes. Sun et al. [31] propose a distinction between template-based and appearance-based methods. Template-based methods use more classical concepts of correlation between the sub-image containing the candidate sample and a template image. Template images are usually built to extrapolate (through the correlation operator) visual information which is supposed to be typical of the object under consideration. Examples could be the licence plate or the characteristic U shape of the rear of a vehicle.

Appearance methods rely on techniques that are more popular today such as Support Vector Machines (SVM) and Neural Networks (NN). In particular, the HV happens through a classifier which requires an off-line training to precede the on-line testing stage. The classifier can be trained with explicit (e.g. edges, lines, corners) or implicit (Haar, Histogram of Gradients (HOG), Scale Invariant Feature Transform (SIFT), Gabor Filters) features.

In our application we are particularly concerned with the detection of the rear of vehicles. This is because the head mounted camera will be obviously pointing towards the front of the car. According to Antony and Suchetha [2], the most effective classifier for detecting rear faces of vehicles is the Haar+Adaboost classifier. This classifier combines the outcomes of multiple decision trees of depth 1 in a weighted manner. Weightings are chosen by a boosting algorithm during training as explained in [16].

Pedestrian Detection

Pedestrian detection has also been object of much research in the past decades. An exhaustive review of existing methods is presented by Dollar et al. [6] from Caltech in their paper *Pedestrian Detection: An Evaluation of the State of the Art*. Here the authors, focusing on sliding window approaches, point to the HOG based feature extraction as one of the most effective means of pedestrian detection.

Even though they claim that no single-feature extraction has been shown to outperform HOG, complementary approaches to HOG exist which can boost the detectors' performance: in the same paper multi-feature approaches, colour self-similarity and motion are considered and reviewed.

Finally, according to the authors, despite the variety of methods proposed throughout the years to improve pedestrian detectors, much work has still to be done in this area to make them robust in more critical situations. As an example the paper illustrates how severely the performance of common detectors degrades with partial occlusion of the test samples. This clearly imposes a significant limitation on our device's performance as partially occluded pedestrian could represent a serious danger for the driver.

Multi-class Object Detection

Given that we are actually after the detection of several types of objects, it may be convenient to use a single multi-class object detector instead of multiple single class detectors like those discussed so far. In particular, advanced machine learning methods have been proposed in the last few years to achieve fast and robust object classification from images. Below we mention two popular ones.

Random Forest The random forest is a rather popular machine learning technique that can be used for multi-class detection. It is an ensemble of independently trained decision trees. Randomness is injected into the tree structures at training stage with the effect of increasing robustness by preventing over-fitting [4].

Neural Networks Neural Networks (NN) have become a leading machine learning tool and have shown extremely good performance in real time object detectors thanks to architecture such as You Only Look Once (YOLO). The drawback is certainly their complexity and very long training times; however pre-trained models are widely available which makes NN a very good option for our system [22].

Comments

Our research illustrated that the literature is very rich of object detection methods; moreover, differently from lane detectors, object detectors usually do not make any assumptions on the camera mounting.

Even though the selection of a region of interest is not strictly necessary, we will show throughout this report that it can still be desirable to restrict the region of the image where objects are searched for in order to both improve the effectiveness and the computational efficiency of detectors.

Clearly, as before, such restriction of the search region can only be obtained with an accurate head tracking in place.

2.2 Head Tracking

Our initial research into road feature detection highlighted the need for an accurate extraction of a region of interest from the input frame. Since the camera is head mounted and hence free to move, it was also clear that the selection of the ROI had to be performed dynamically.

Therefore in this section we focus on the implications of such unbonded camera (Figure 2.6) and more specifically on the topic of head tracking aimed at supporting the required dynamic ROI selection. What we refer to as head tracking, in Jargon is known as camera pose estimation (Figure 2.7), that is to say the estimation of the 6 DOFs combination of camera position (3 DOFs) and orientation (or attitude, 3 DOFs).

(a) Cockpit perspective in normal conditions

(b) Cockpit perspective with head rotation

Figure 2.6: Illustration of the variation of perspective correspondent to driver's head movements.

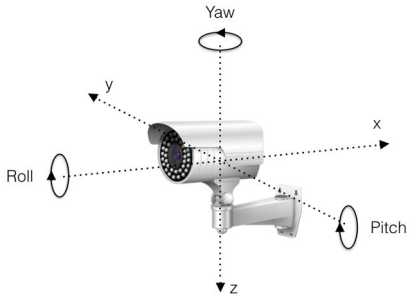


Figure 2.7: Illustration of the 6 DOFs camera pose.

Position and attitude determination are a well solved problem on large scales (e.g. airplanes navigation). However, when moving to smaller scale problems, a few issues kick in which severely complicate things. Therefore, in this section we will discuss possible ways of facing those issues and successfully determining the pose of the driver's head.

2.2.1 Attitude

Inertial Measurement Unit

When it comes to attitude determination, Inertial Measurement Units (IMUs) are usually the preferred devices. However, in order to be able to calculate the relative movement of the driver's head w.r.t. the cockpit using a single IMU, the car would need to be an inertial reference system, which is not the case (i.e. it is subject to external forces[2]).

In particular, because of the underlying car linear and rotational accelerations, the relative angular movement of the drivers' head with respect to the car's reference system (which we are interested in) is the result of its absolute angular

[2]Newton's second law: unbalanced forces cause the car to accelerate, hence making it a non inertial frame of reference.

movement (i.e. w.r.t. to the Earth reference system) minus the car's absolute angular movement (again, w.r.t. the Earth reference system).

It turns out that accurate measurements of the first quantity are not easy to achieve: all inertial sensors suffer from the underlying movement of the car; gyroscopes have very high drifts and are impractical for steady state attitude measurements [37]; magnetometers suffer from both hard and soft magnetic interference due mainly to the metal frame of the car [21].

On top of this, even if there were a way of accurately estimate the first quantity, a direct measurement of the car's movement w.r.t. the Earth reference system would be impossible as by requirements we are not allowed to install any device onto the car nor make use of any on-board system that might hold that information.

It follows that our system cannot rely uniquely on IMUs but rather it needs some other way of improving the reliability and stability of the attitude measurement.

Vision-based Solution Fortunately, inertial measurements are not the only way that has been proposed for attitude determination. In particular in the aerospace industry, which is where most of the literature about pose determination has been produced since the 50's [29], several alternative approaches exist.

For example, Santos et al. in [29] combine a gyroscope with a downward pointing camera with landmarks in the field of view for the attitude estimation of a drone. Additional vision-based techniques are discussed in [8] and [3] in the context of airplane and satellite navigation.

However, in [29] the way landmarks are identified and tracked is not discussed while in [8] and [3] the vision problem is significantly easier than ours. It follows that in order to make use of the results obtained in these papers we will have to design and implement our own cockpit feature identification and tracking.

2.2.2 Position

The limited accuracy of GPS data [11] makes its use impractical in small environments. It follows that to calculate the Cartesian position of the head inside the cockpit we will have to rely on some other technology. Once again it could be possible to use vision based approaches: distance measurements could be achieved with stereoscopic vision or with perspective estimation.

2.2.3 Pose

So far we have considered the problems of attitude determination and positioning separately. In actual facts, methods to obtain a simultaneous estimate of both (often referred as the *pose* of the camera) exist as a result of the research in Simultaneous Location And Mapping (SLAM).

One of such method is known as Projective-n-Points (PnP) which exclusively uses vision although it can be combined with an IMU through a Kalman Filter [26]. Such method uses an input frame as well as a pre-computed 3D map of the environment to determine the 6 DOFs pose of the camera; algorithms exist to perform this estimation very efficiently [18].

The main disadvantage of PnP is that it requires a number of expensive resources such as a depth camera or radar paired with a standard optical camera to build the off-line 3D map of the environment which the algorithm relies on. Furthermore in some cases, in order to achieve a satisfactory accuracy, markers should be used, which would be against requirements.

In order to be able to complete the development of the entire system on time, we decided not to risk the implementation of PnP; nevertheless, in this report we include some preliminary investigations we conducted on it and, most importantly, we strongly suggest further research in the applications of this algorithm in our system as part of future developments of the project.

2.3 Review of Commercial AR Devices

In this section we provide a brief analysis of commercial AR devices. In particular, the knowledge of their hardware specifications will drive some of our design choices.

2.3.1 Microsoft HoloLens

One of the most recent AR Devices to become available to the public was Microsoft HoloLens. The set of specifications is provided in Table 2.1.

Table 2.1: Table of Hardware Specifications for the Microsoft HoloLens [36].

Microsoft HoloLens: Specifications		
Hardware	Processor	Custom Microsoft Holographic Processing Unit HPU 1.0 Intel 32-bit architecture
	RAM	2GB
	Memory	64GB Storage
Optics	Display	See-through holographic lenses (waveguides) 2x HD 16:9 Automatic pupillary distance calibration 2.3M total light points holographic resolution 2.5k light points per radian
Sensors	IMU	9 axis (Accelerometer, Magnetometer, Gyroscope)
	Cameras	4x environment understanding cameras
	Microphones	4x
	Ambient Light	
	Main Camera	2MP photos, HD video
Communications	GNSS	GPS/GLONASS
	Wi-Fi	802.11ac
	Bluetooth	Bluetooth 4.1 (HS, BLE, ANT+)

2.3.2 ODG R7 Smart Glasses

ODG is a leader in AR Glasses with some of the best reviewed headsets. The specifications for the R7 model which is available to the public are shown in Table 2.2.

Table 2.2: Table of Hardware Specifications for the ODG R7 Smart Glasses [36].

ODG R7: Specifications		
Hardware	Processor	Qualcomm SnapdragonTM 805 2.7GHz quad-core Processor
	RAM	3GB Pop LP-DDR3 RAM
	Memory	64GB Storage
Optics	Display	Dual 720p Stereoscopic See-thru displays at up to 80fps
	Transparency	80% See-thru Transmission
Sensors	IMU	9 axis (Accelerometer, Magnetometer, Gyroscope)
	Altitude	
	Humidity	
	Ambient Light	
Communications	GNSS	GPS/GLONASS
	Wi-Fi	802.11ac
	Bluetooth	Bluetooth 4.1 (HS, BLE, ANT+)

2.3.3 Comments

It is clear by looking at the technical specifications that in terms of hardware commercial AR devices are not much different from a top end smart-phone. This is especially true as far as connectivity and motion sensors are concerned. Only some devices, such as the very new Microsoft Hololens, have some differences in the dedicated holographic processing unit and in the multiple cameras, which may indeed make some tasks easier in the creation of AR content.

2.4 Comments

Our findings highlighted that although much research has been conducted in the field of computer vision for driving applications, very little has been done using unbonded cameras.

In particular the topic of driver's head tracking is rather new in the field and it is therefore unsurprising that, just to make an example, most lane detection methods make several assumptions on the camera mounting position.

Therefore, in designing our AR system we dedicated particular attention to the development of an algorithm that is able to track and compensate for the head movements, hence bridging the gaps between an unbonded camera and existing state of the art CV methods.

Chapter 3

System Design

3.1 Introduction

If a steady camera were assumed, the exhaustive literature review presented in the previous chapter would allow us to design a rather standard driver assistance system.

Figure 3.1: Steady camera based system high level diagram.

The diagram in Figure 3.1 shows that a train of video frames is the input to the system; for every frame, a predetermined region of interest (ROI) is cropped out; the sub-image is then processed in order to extract road features and generate alerts to the driver; finally, alerts are presented by means of a user interface (in our case AR-based).

Key

It should be obvious by now that the kind of system presented in Figure 3.1 is not able to deal with a moving camera and will therefore fail as the driver looks around the cockpit (e.g. to look at the side mirrors). In order to make the system tolerant to camera movements, we modify it as shown in Figure 3.2.

Again, a train of frames is the input to the system; this time, however the frames go through a pre-processing stage that estimate the camera pose and

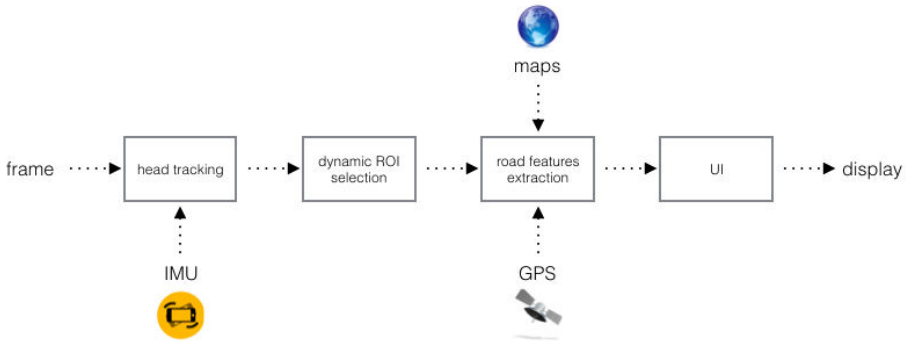


Figure 3.2: Modified system high level diagram. The selection of the Region of Interest (ROI) is performed dynamically thanks to the preceding head tracking stage.

compensate any movement by dynamically selecting the region of interest; only once the frames have been pre-processed, can they go through similar stages as in Figure 3.1. Note furthermore that we have considered a more advanced feature extraction which takes into account the road shape (from maps) and the position of the car (from GPS).

In this chapter we are going to present the results of the investigations we conducted on each of the subsystems illustrated in Figure 3.2. We will follow the same order as the diagram: we will start with a brief discussion on data acquisition; we will then move to the problems of determining the device pose and hence finding the ROI in the camera frame; after that we will discuss methods to detect road features; finally we will propose informative but non-invasive ways of displaying AR content to the driver [5].

3.2 Data Acquisition

The complete absence of a data set with videos taken from the inside of the car cockpit forced us to drive around public roads to collect our own data. In order to capture the data necessary to our research an ad-hoc iPhone application was developed. A full description is provided in Section 6.2. The source code as well as the data collected is publicly available through the project repository (see Appendix A and Appendix B).

The close similarities between common AR devices and currently available smart-phones, make the latter a desirable platform for evaluating the feasibility and the performance of the system under development. Furthermore, thanks to third party accessories, smart-phones can be head-mounted and therefore

emulate the much more expensive currently available AR devices. Furthermore, due to their very early stage of development, we regarded driving with AR headsets on as too risky for our own safety.

3.2.1 Data Set

Out of the several hours of data collection, we selected the most representative video shootings to form two versions of the data set. The first, older version is taken with a steady camera mounted on the passenger seat; the second, most recent version is taken with the camera mounted on the driver's head and is therefore the most realistic. Nevertheless, both datasets turned out to be very useful at different stages of the project.

Version 1 - Steady Camera

Table 3.1 summarises the main features of the first version of our dataset.

Table 3.1: Table of specifications of the steady camera dataset.

Feature		Comments
Mounting	Static	Passenger Seat
Road Scene	/	Highway
Camera	✓	640×480 $30fps$
IMU	$\sim$	3-axis (gyroscope only) $100Hz$
GPS	✓	<5m

Version 2 - Moving Camera

Table 3.2 summarises the main features of the second version of our dataset.

Table 3.2: Table of specifications of the moving camera dataset.

Feature		Comments
Mounting	Moving	Head Mounted
Road Scene	/	Highway
Camera	✓	640×480 $30fps$
IMU	✓	9-axis $100Hz$
GPS	✓	<5m

3.3 Pose Determination

In Section 2.2 we discussed possible ways of estimating the camera pose inside the car cockpit using information from the camera itself and from an Inertial Measurement Unit (IMU). In particular we made two key observations:

- The IMU is likely to suffer from drift effects and interference due to the underlying movement of the car;
- Purely vision based attitude determination methods could be too slow to run in real time.

Keeping in mind the two observations, in this section we would like to propose a method that is mainly vision based but that is able to exploit IMU data (when available) to make the estimate faster and more robust.

Throughout this section we will use 1200 frames from a video belonging to our dataset which are dense of head movements. The ground truth for the attitude of the driver's head has been manually labelled. This is illustrated in Figure 3.3.

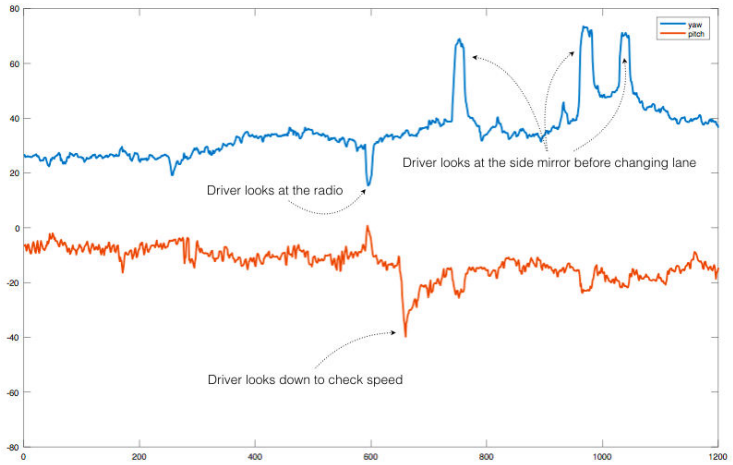


Figure 3.3: Manually labelled reference for the pitch and yaw of the driver's head over time. Frame 17400 to 18400 of the moving camera dataset.

Key

To simplify the problem while still being able to meet requirements, we make the reasonable assumption that the driver Cartesian position remains fixed and that only 3D rotations are allowed.

This lets us reduce the original 3D problem into a 2D problem as the 3D model of the car cockpit can now be projected onto a 2D surface. The 2D projection can be obtained by merging multiple photos into a panoramic view.

To build the panoramic image used for head tracking, we took seven images of the car cockpit with at least 50% overlap (Figure 3.4) and merged them

using the Photoshop CS6 *Automerge* command to create the desired output (Figure 3.5).

(a) (b) (c) (d) (e) (f) (g)

Figure 3.4: Collection of cockpit images taken in a range of driver's head orientations.

Figure 3.5: Resulting panoramic image.

A careful observer would note that a slight distortion is introduced when the panoramic image is created; one possible way of avoiding such distortion would be to use the raw images directly into the head tracking algorithm proposed which could be done with minor modifications.

However, as we will soon demonstrate, such distortion has very little effect on the accuracy of the head tracker. Moreover, the drawback of using multiple images instead of one merged panoramic photo would be that each image would need its own careful labelling which is necessary in order to both be able to detect road features and to display AR content to the driver.

Key

Once the panoramic image has been computed (off-line), the camera pose can be calculated from the rigid transform that matches the input frames with the panoramic view itself (on-line). Figure 3.6 gives an intuitive demonstration of this idea.

Figure 3.6: Simple illustration of the rigid transform that maps the input frames (left) to the reference image (right). Through this mapping, a measure of the camera attitude can be estimated.

In what follows we present our findings in chronological order: our first version of the head tracker used the principle of cross correlation between images to estimate the rigid transformation; our second and more advanced version uses local feature matching and a RANSAC based transformation estimation.

3.3.1 Naive Head Tracking

With our first version of video stabilisation we managed to track the driver's head attitude at very high speed combining a vision based approach with the gyroscope data. In order to achieve a very desirable computational performance we made one additional assumptions: of the driver's head attitude we retained yaw and pitch but we neglected the roll.

This assumption is supported by the observation that the driver is very likely to look at his sides (e.g. side mirrors, radio, etc.) as well as up and down (e.g. to check the speed); however rotations on the roll axis are much less frequent and usually have a negligible amplitude.

Algorithm To start with, let us consider the very simple algorithm:

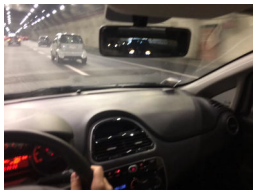
Data: Reference Frame, Input Frame
Result: Attitude Estimate
read Reference Frame;
for *each Input Frame* **do**
  read Input Frame;
  find the traslation that yields maximum correlation between Reference Image and Input Image;
end

Algorithm 1: First attempt.

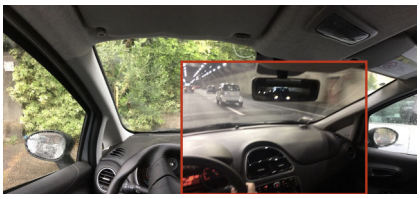
The method is based on the principle of cross correlation, which means that the input frame is compared against a reference image by means of a sliding window until the maximum correlation point is found. Figure 3.7 shows an

example pair of input image and reference image as well as the desired algorithm output.

(a) Template Image



(b) Input Frame



(c) Max Correlation

Figure 3.7: Illustration of the sliding window approach to find the position yielding maximum correlation between the input frame and the template image.

Local Optimisation In practice, however, this method has a major drawback, that is that it is computationally unsustainable. In fact searching for the globally optimal solution of the problem described is extremely expensive: assuming a full resolution 1200×500 panoramic image and a 640×480 input frame, the number of multiply-accumulate operations would be in the order of $\approx 10^{11}$; $\times 3$ if colour images are used.

In order to overcome this problem, images are scaled down in size by a factor of 5 and only a local search is performed. Furthermore, the local minimum is found by the coordinate direction method rather than through an exhaustive search of the neighbourhood. This is illustrated in Figure 3.8.

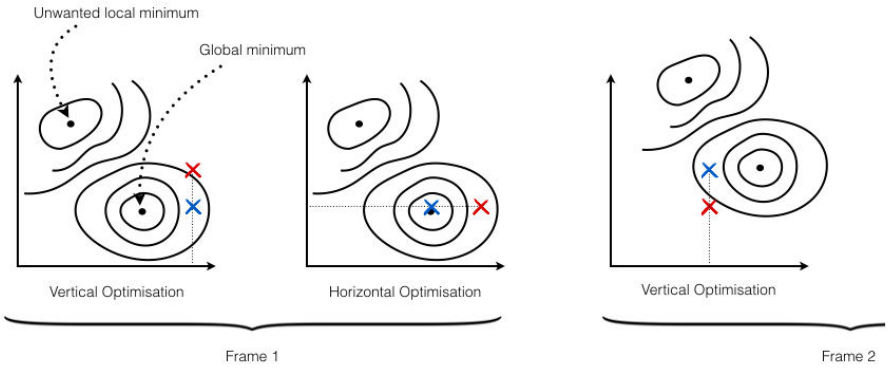
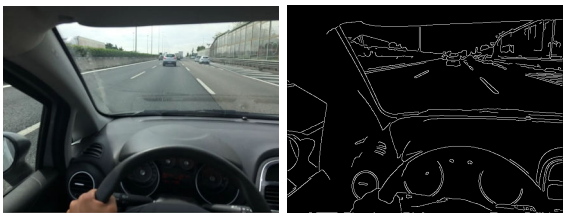


Figure 3.8: Illustration of the iterative coordinate direction method. At each iteration, the starting point is drawn in red while the landing point is drawn in blue.

Clearly, such a method is not guaranteed to converge; nevertheless, it has been observed experimentally that if the neighbour is made large enough the method becomes stable when applied to our dataset. Typically, at $30fps$, a local search of 15 pixels horizontally and 15 pixels vertically is enough to reliably track the driver's head rotations.

Edge Detection An additional limitation comes from the fact that 3-channel images would be very sensitive to the scene (i.e. light conditions, shadows, etc.), which is undesirable in a constantly changing environment like ours: to make the method more robust, binary edge detected images are used instead of 3-channel colour images (Figure 3.9). Note that binary images also contribute to a reduction in computational requirements.



(a) Daylight (b) Canny Edge

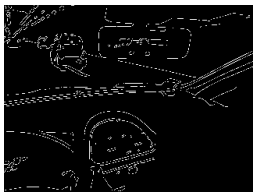
(c) Tunnel (d) Canny Edge

Figure 3.9: Illustration of the invariance of edge detected images to most scene variation (e.g. light, background, etc.).

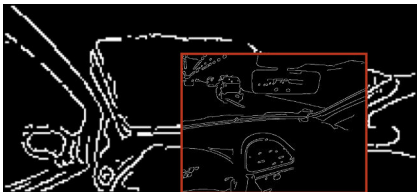
Robustness The vision based approach to head tracking that we are discussing works by matching patterns between the input image and the reference image. However, within the input and reference images, only those patterns that belong to the car cockpit should be tracked and anything outside the (transparent) wind-shield should be ignored.

To this purpose we manually removed all unnecessary features from the edge detected reference panoramic image (Figure 3.10). Removing the unnecessary features from the input image cannot be done with the same level of accuracy because the rigid transform between the input image and the reference image is unknown (in fact it is to be estimated).

(a) Template Image



(b) Input Frame



(c) Max Correlation

Figure 3.10: Illustration of the sliding window approach to find the maximum correlation between the (binary) input frame and the template image. Non-car features have been manually removed from (a).

Nevertheless, by noting that such transform cannot drastically change from one frame to another ($30fps$), one could decide to rely on the previous frame's attitude estimate in order to approximately remove the unwanted features visible outside the transparent wind-shield. Clearly the price to pay is that because of this approximation few desirable features will be removed instead of some others unwanted features and vice versa.

As a result of these few remarks, we list the modified version of the algorithm:

Data: Reference Frame, Input Frame, N_Iter, Search Region
Result: Attitude Estimate
read Reference Frame;
delete unnecessary regions from Reference Frame;
edge detect Reference Frame;
for *each Input Frame* **do**
    read Input Frame;
    repeat
        find the vertical traslation (within Search Region) that yields maximum correlation between Reference Image and Input Image;
        find the horisontal traslation (within Search Region) that yields maximum correlation between Reference Image and Input Image;
    until *N_Iter times*;
end

Algorithm 2: Improved algorithm using local optimisation and edge detected images. Note that the inner loop can be repeated an arbitrary number of times. The more it is repeated, the closer the solution will be to the maximum correlation point (Figure 3.8).

Performance Figure 3.11 shows that the proposed algorithm is able to track the head movements although some noise is visible at the output. The RMS of the error for the pitch and the yaw is shown in Table 3.3.

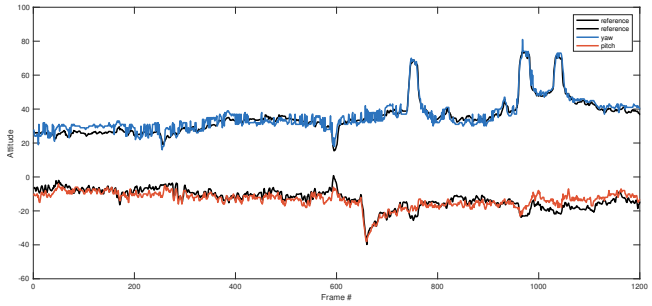


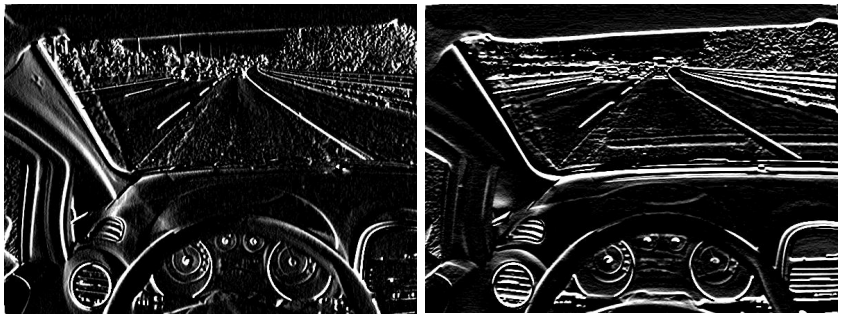
Figure 3.11: Performance of the correlation based method against the reference.

Table 3.3: RMS of the error in the correlation based method.

	Correlation (15x15 pixels search region)
Yaw Error (RMS)	3.13
Pitch Error (RMS)	4.28

Unimplemented Enhancements Here we propose a few enhancements that could improve the performance of the algorithm.

Firstly, instead of edge detection, directional image derivatives could be used to form two pairs of reference and input images. The pair obtained by taking horizontal derivatives should be used for the horizontal matching and the pair obtained by taking vertical derivatives should be used for vertical matching. This enhancement is likely to reduce noise in the coordinate direction optimisation.



(a) Original Frame (b) Cropped Frame

Figure 3.12: Example of directional image derivatives applied to the input frame.

Another enhancement comes from the observation that only the central region of the input frame can be reliably matched to the panoramic image while the edges (and especially the corners) are not well matched due to non-linear artefacts introduced in the construction of the panoramic image itself. Therefore it is suggested to crop the central region of the input frame to improve robustness.

Improving the Performance with an IMU

As previously discussed, the performance of the system can be improved with the addition of an Inertial Measurement Unit (IMU), provided that a few limitations of the IMU are overcome, namely the steady state error (drift) and the interference due to the underlying movement of the car.

Implementation Of the inertial measurement unit, which usually includes a magnetometer, an accelerometer and a gyroscope, we actually make use of the gyroscope only, confirming the assumption that the Cartesian position of the head inside the cockpit remains fixed and that the rotation around the roll axis is negligible.

We start by analysing the output traces correspondent to the yaw and the pitch. What the raw output of the gyroscope represents is the rate of change of

the angle (or angular velocity). Therefore, in order to obtain the angle we just have to perform the following integral:

$$\theta(t) = \int_0^t \omega(t)dt \tag{3.1}$$

Obtaining the following trace for the angle against time.

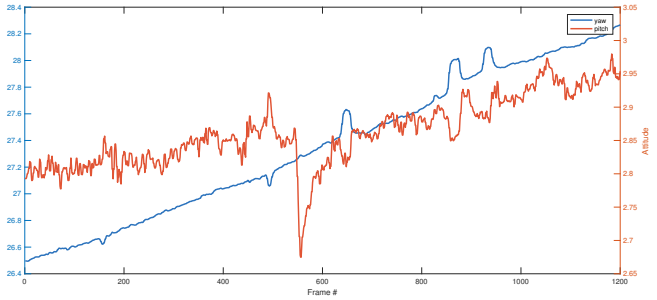


Figure 3.13: Output of the gyroscope after integration block.

The result of the integration looks far from being correct: in particular the signal we are after sits on top of a very significant trend. This effect, known as *drift*, is due to the small bias that characterises the output of any MicroElectroMechanical Systems (MEMS) gyroscope [37].

The gyroscope drift can be compensated by de-trending (essentially high-pass filtering) the integrator output; however, achieving a perfect correction is highly unlikely and for this very reason, as already discussed, a vision based attitude determination stage is unavoidable.

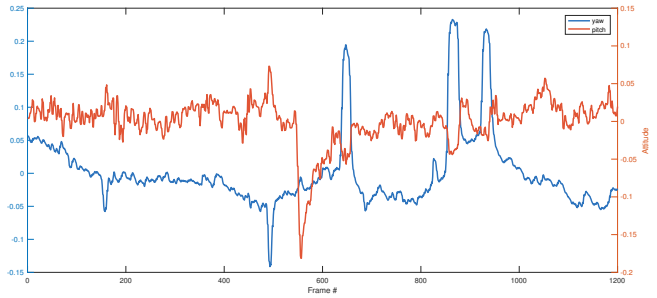


Figure 3.14: De-trended gyroscope output.

Finally, in order to convert the angle to a shift in pixels we need to take into account the internal camera parameters, that is to say the focal length and the

size of the CCD, as shown mathematically in the two equations below:

$$t_x = f \cdot \tan(\theta_x) \cdot \frac{w_{frame}}{w_{CCD}} \tag{3.2}$$

$$t_y = f \cdot \tan(\theta_y) \cdot \frac{h_{frame}}{h_{CCD}} \tag{3.3}$$

Where t_x and t_y are the resulting shifts in pixels; f is the camera focal length; θ_x and θ_y are the yaw and pitch angle respectively; w and h are the width and the hight of the frame (in pixels) and the CCD (in meters) according to the subscript.

Figure 3.15 shows a graphical representation which can be used to derive (3.2) and (3.3).

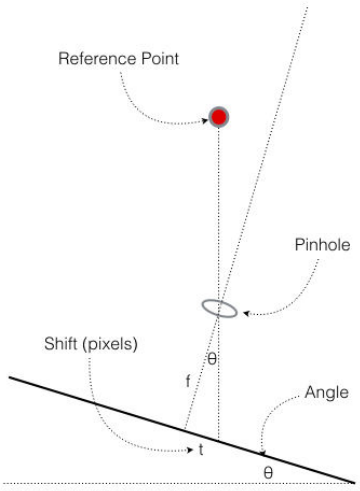


Figure 3.15: Graphical representation of the conversion between attitude angles and pixel shifts.

Once the pitch and yaw are obtained in pixels, they can be used for tracking purposes. In particular the gyroscope can be used first to obtain a candidate attitude measure that is then refined by the vision-based algorithm. Since a rough estimate is pre-computed, the search region of the vision algorithm can be reduced to focus on a finer search which can significantly improve performance.

By combining the IMU with the vision-based method we managed to fix the steady state error of the device. However, the interference due to the underlying car movement remains unsolved. Here we list a few viable ways of tackling the problem.

First let us consider an high way driving situation. Furthermore let us consider the steepest curve on the Italian A1 highway which has a radius of curvature of $\approx 442m$ and speed limit at $100km/h$ [27]. Simple calculations show that the angular velocity is:

$$\omega = \frac{v}{r} \approx 0.06 rads/s \tag{3.4}$$

or $\approx 3.6^o/s$; where v is the tangential velocity and r the radius of curvature. At $30fps$ the interference due to the gyroscope is likely to be negligible ($\approx 0.11^\circ$ per frame).

In urban environments, the gyroscope interference due to underlying car movements could be higher, in which case a map together with accurate GPS positioning could be used to either disable or compensate the gyroscope errors on steep curves. Even better would be to have one gyroscope on the head and one on the body in order to produce a much more accurate differential measurement and hence reject the common mode signal due to car movements. This however, may conflict with the design requirements.

Performance It is interesting to note that both stand alone vision based and IMU based approaches have their own limitations. However, when the two methods are combined, they yield an attitude compensator with very desirable properties. Figure 3.16 and Table 3.4 show the improvement in system performance with the addition of the IMU in the system.

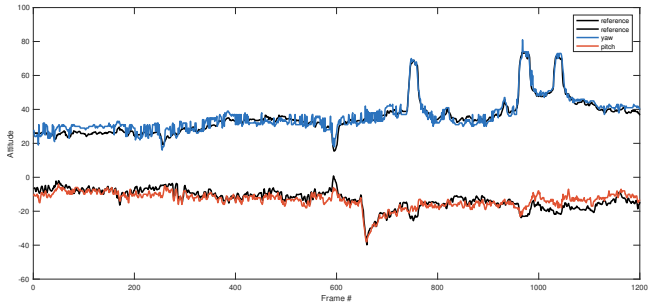
Table 3.4: RMS of the error in the correlation based method with and without IMU.

	Correlation (15x15)	Correlation (5x5 + IMU)
Yaw Error (RMS)	3.13	2.8
Pitch Error (RMS)	4.28	4.0

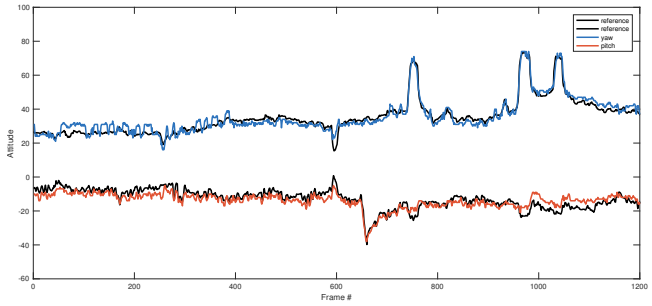
3.3.2 Robust Head Tracking

The second version of the pose estimation algorithm aims at retaining the good tracking performance of the previous method while rejecting the unwanted noise seen at the system output. To do so we use local feature matching instead of image correlations through which we estimate a (simplified) homography transformation between the input image and the reference image.

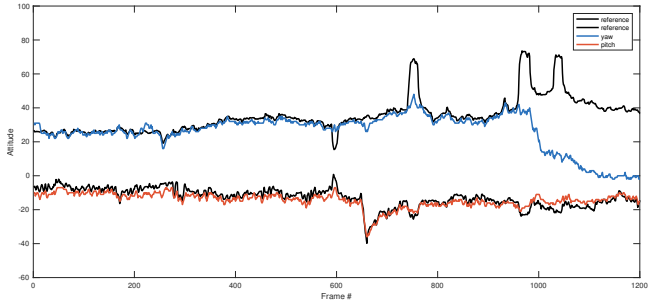
This method is a result of a number of MATLAB experiments: we tried a variety of feature descriptors (i.e. SIFT, SURF, ORB, etc.); we implemented our Greedy Nearest Neighbour (NN) algorithm for feature matching and an efficient



(a) Coordinate direction (15×15 search region)



(b) Coordinate direction (5×5 search region) + IMU



(c) Coordinate direction (5×5 search region)

Figure 3.16: Performance of the correlation method (a) with no IMU and larger search region, (b) with IMU and smaller search region, (c) with no IMU and smaller search region. Note that in the latter case the system is not able to track the head movements.

homography estimation algorithm which uses a RANdom SAmple Consensus (RANSAC) method.

Note that, for a fair comparison, the reference image we used was the same as in the previous algorithm.

Algorithm Given the input frame and the reference panoramic image, we can list the new algorithm for attitude determination as follows:

Data: Reference Frame, Input Frame
Result: Attitude Estimate
read Reference Frame;
extract features from Reference Frame;
for *each Input Frame* **do**
    read Input Frame;
    extract features from Input Frame;
    match features using NN algorithm;
    estimate homography;
end

Algorithm 3: First attempt.

Feature Extraction By feature extraction we mean the process of detecting interest points in the image and calculating descriptors which ideally let us identify any of the chosen interest points uniquely. These descriptors can then be used to match the same features appearing in two distinct images. Examples of detected Speeded Up Robust Features (SURF)[1] in the reference and input image are shown in Figure 3.17.

(a) Reference Image

(b) Input Frame

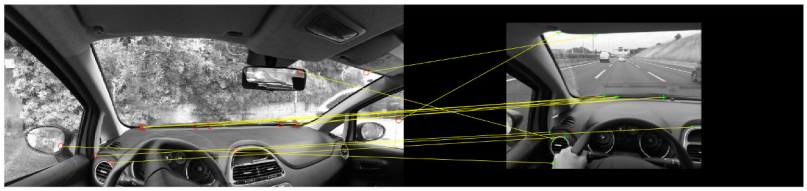
Figure 3.17: Detected SURF features in (a) reference image and (b) input frame.

Feature Matching Feature matching is done using a computationally efficient Nearest Neighbour (NN) algorithm. The input to the algorithm are two

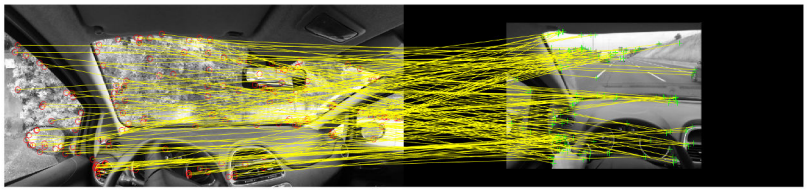
[1] Computationally efficient variant of the Scale Invariant Feature Transform (SIFT)

sets of image descriptors (one from the input frame and one for the reference panoramic image); the algorithm is Greedy as it goes through one of the two input sets and at each step it finds the best possible match in the other descriptor set; distance between descriptors is in terms of Euclidean distance; furthermore a threshold is in place so that features that have the best match but whose distance is still above threshold are left unmatched.

Figure 3.18 illustrates the benefits of a threshold in the NN algorithm. It is in fact clear in Figure 3.18b that the absence of threshold adds a significant amount of outliers.



(a) Reference Image



(b) Input Frame

Figure 3.18: Matched features using NN algorithm.

Homography Matrix Estimation Under the assumptions stated, the Homography matrix reduces to:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

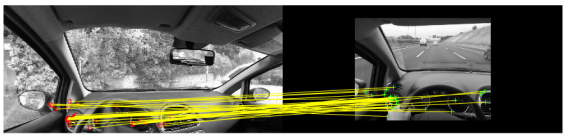
Or to:

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) & t_x \\ sin(\alpha) & cos(\alpha) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

if one wants to allow for rotation due to the driver's head roll.

Note that (3.6) is the transformation matrix of a rigid transform while (3.5), by neglecting the rotation on the roll axis, intrinsically makes the same assumptions as in the correlation method described in the previous section. For a fair comparison between the two methods we estimate the transformation as in (3.5), therefore neglecting once again rotations on the roll axis.

Even under the assumptions stated, an accurate measure of the homography matrix is not straightforward. In particular, given a list of correspondences between the two images, we found that a simple least squares solution does not give satisfactory results. Instead RANSAC should be used to detect and eliminate outliers. Figure 3.19 shows the homography estimate using least squares and RANSAC approaches. The re-projection error is summarised in Table 3.5.



(a) Matched Features

(b) Least Squares: RE = 124.72

(c) RANSAC: RE = 11.26

Figure 3.19: Comparison of homography estimation using (a) least squares and (b) RANSAC.

Table 3.5: Re-projection error of LS and RANSAC calculated as the mean square distance (in pixels) between the original feature points in the input frame and the projection of their correspondent feature points from the reference image.

	LS	RANSAC
Re-projection Error	124.72	11.26

Figure 3.20 shows the resulting head tracking performance. In particular while the system appears to be able to track the changes in head orientation, a significant noise seems to affect the output. In order to try to reject such noise, we propose a few additional modifications.

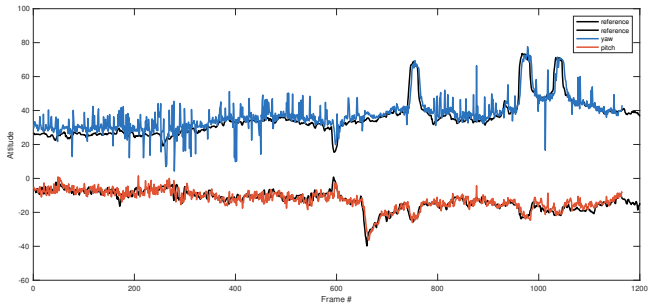


Figure 3.20: Performance of the local feature matching method.

Modification of the Reference Panoramic Image As before, a significant portion of the reference image contains unnecessary features which are likely to cause errors and do not aid the homography estimation. Therefore it is desirable to remove those portion of reference image and to leave only areas which are dense of features belonging to the car cockpit.

Furthermore, as discussed previously for the correlation-based method, using the assumption that the transformation does not significantly change from one frame to another, we can use the homography estimated in the previous frame to remove unwanted portions of the input image as well.

The final algorithm is the result of all the observation considered:

Data: Reference Frame, Input Frame
Result: Attitude Estimate
read Reference Frame;
extract features from Reference Frame;
for *each Input Frame* **do**
    read Input Frame;
    if *is not first frame* **then**
        get previous frame Homography matrix;
        crop Input Frame to isolate cockpit features;
        crop Reference Frame to restrict the field of view to be the same as Input Frame;
    end
    extract features from Reference Frame;
    extract features from Input Frame;
    match features using NN algorithm;
    estimate homography;
end

Algorithm 4: Modification of the algorithm to remove non-cockpit feature points before matching features.

The benefit of this modification can be appreciated by looking at the performance of the method over a number of video frames. Figure 3.21 shows the significant improvement in feature matching after the latter modification while Figure 3.22 demonstrates a significant output noise reduction.

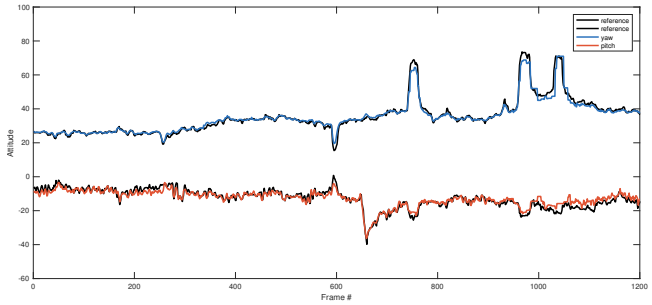


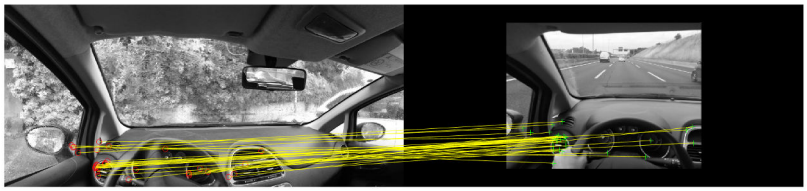
Figure 3.22: Performance of the feature matching method after non-cockpit feature points are removed.

IMU

Figure 3.22 shows that the improvements discussed are able to significantly stabilise the output of the algorithm by preventing a large number of matching errors. Unfortunately however, the new algorithm is less responsive to sudden

(a) Reference Image

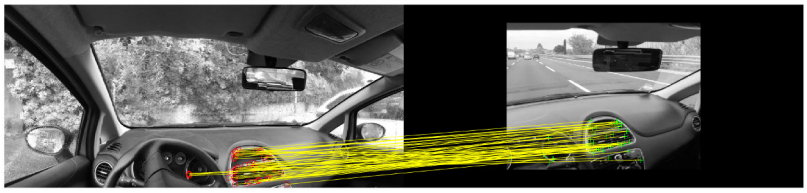
(b) Input Frame



(c) Matched Features

(d) Reference Image

(e) Input Frame



(f) Matched Features

Figure 3.21: Feature matching after non-cockpit feature points are removed.

changes in head pose as shown around frame #1000. This is most probably due to the failure of the assumption that the variation in homography matrix across two consecutive frames is quasi-zero.

We want to show that this problem can be conveniently solved with an IMU which is able to recover the responsiveness of the tracking at almost zero computational cost. The result of such further modification is shown in Figure 3.23.

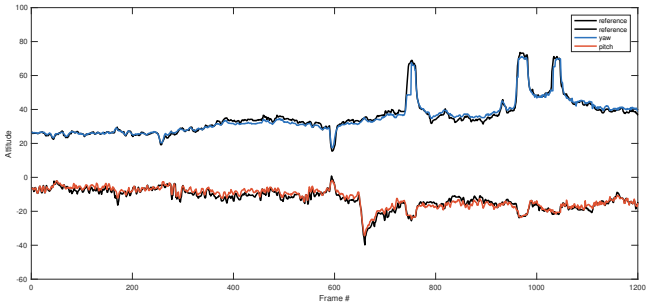


Figure 3.23: Performance of the feature matching method in combination with an IMU.

Table 3.6: Comparison of local feature matching algorithms.

	First Attempt	Modified	Modified + IMU
Yaw Error (RMS)	9.82	2.15	1.62
Pitch Error (RMS)	7.53	1.98	1.54

3.3.3 Comments

Both the implementations exhibited a generally desirable head tracking performance.

The correlation based method was definitely the least accurate and stable of the two, but it demonstrated a significantly better computational performance (above $100fps$) as well as desirable robustness to changes in light intensity.

The local feature matching method on the other hand was very accurate and stable in normal conditions although the feature extraction and matching stages made the algorithm the slowest of the two. Furthermore we observed a degrade in performance under significant changes in illumination (e.g. tunnel, night driving, etc.).

It follows that no algorithm was by far superior and that their choice may depend on the specific application. It would also be possible to combine the two methods for an even improved head tracking.

Table 3.7: Comparison of correlation based and local feature matching algorithms under constant day light conditions.

	Correlation (5x5 + IMU)	Local Feature Matching (ORB Features + IMU)
Yaw Error (RMS)	2.8	1.62
Pitch Error (RMS)	4.0	1.54

3.4 ROI Extrapolation

Once the head problem has been solved as in 3.3, extracting the ROI becomes very simple. In fact, it is sufficient to choose the Region Of Interest in the reference panoramic image and then apply the estimated transformation to obtain the coordinates of the ROI in the input frame. We regard the choice of the ROI in the reference image as *labelling*. Figure 3.24 shows an example of labelled ROI and the corresponding region extracted from the input frame. Figure 3.25 shows a robust ROI extraction under sever head rotations.

Key

It is worth stressing once again that this novel dynamic ROI selection is the key to bridge the gaps between a moving camera and CV methods aimed at road object detection developed for static cameras. Therefore, we regard this result as one of the most important achievements of our project.

(a) ROI Labelling

(b) ROI Extraction

Figure 3.24: Example of ROI labelling and extraction.

Furthermore, as we will see throughout the next sections, the importance of the dynamic extraction of regions of the input image is not limited to the lane and vehicle detectors but rather it plays a crucial role in populating the driver's AR experience with 3D content. More on this will be discussed in 3.6.

(a) Straight

(b) Straight (ROI)

(c) Down

(d) Down (ROI)

(e) Left

(f) Left (ROI)

Figure 3.25: Example of ROI extraction under severe driver's head movements.

3.5 Road Features Extraction

In this section we would like to show that a correct extraction of the region of interest from the input frame immediately enables us to use standard computer vision algorithms to detect road features.

More specifically we will see that in some cases the knowledge of the ROI is critical for the correct functioning of standard algorithms; in some other cases it is not essential but can lead to a significant improvement in computational performance.

Below we will focus on the detection of road lanes and of surrounding objects such as cars and pedestrians. Support for many other features could in principle be added to the system.

Note Local feature matching based head tracking is be used throughout this section.

3.5.1 Lane Detection

Assumptions

We argued that most lane detection methods assume that the camera is steadily mounted on the car and therefore were not directly applicable to our dataset. Now, thanks to the head tracking and dynamic ROI selection methods proposed in 3.3 ad 3.4, we have finally been able to bridge the gaps between those methods and our unbonded camera setting.

In particular, in order to show the robustness of our head tracking algorithm, we chose to implement a state of the art lane detection method that makes very strict assumptions on the stability of the region of interest [32].

Candidate Lane Points

We start by dividing the ROI into an upper and a lower region. We use the lower region to detect candidate (straight) lanes. Then, using a river-flow method, we find candidate lane points in the upper ROI (i.e. further away from the car). In what follows we explain the detection steps in detail.

Lower ROI In the lower region, straight lanes are assumed. As we explained in our literature review, this is a reasonable assumption as, even on a curved road, lanes would look straight in the lower ROI. To detect straight lanes in a computationally efficient way we calculate a projective transformation between the driver's view and a bird-eye view as shown in Figure 3.26.

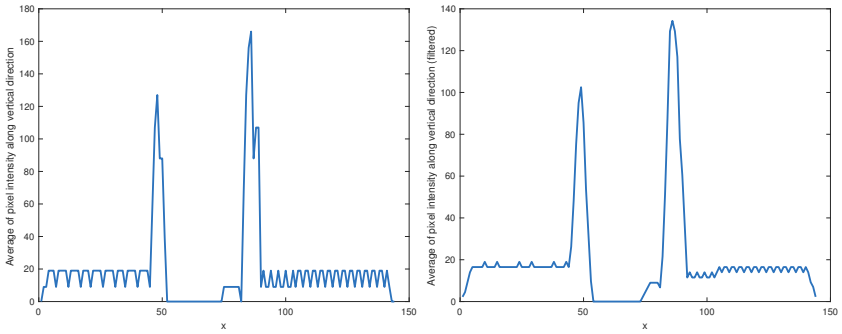
(a) Driver's View (b) Bird-eye View

Figure 3.26: Example of ROI projection from driver's view to bird-eye view followed by directional filtering.

Under the transformation shown, lanes that were previously converging towards the vanishing point appear now straight and parallel. The transformation matrix could be calculated by measuring the height of the camera w.r.t. road level. However, since we know that the Cartesian position of the head will remain fixed throughout the driving session, we can estimate the transformations directly from the frames using manual feature matching.

In the method proposed in [32], directional filters are applied to the frame before the projective transformation in order to highlight the presence of lanes. However, according to the position of the car within the lane, a different orientation of the filters should be picked in order to maximise their effect. Instead we first transform the image and then apply an horizontal directional filter.

Once the image has been correctly projected and filtered, it would be enough to perform a few horizontal scans of the frame to be able to detect the lanes. Instead, in order to increase the robustness of the estimate we perform a vertical sum of the pixels in the image to obtain the signal shown in Figure 3.27a.

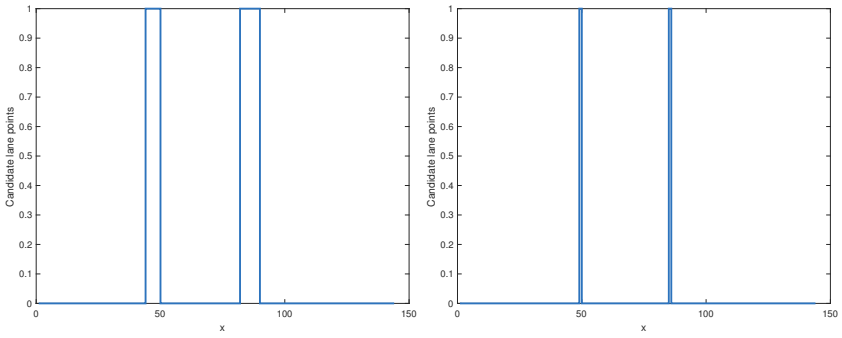


(a) Raw (b) MA Filtered

Figure 3.27: Average of pixel intensities along the vertical direction of the projected ROI; (a) raw and (b) filtered.

The signal is then smoothed with an MA filter (Figure 3.27b) and finally a threshold is applied to identify the horizontal position of lanes in the frame. Note

that the thresholding is modified to reject non local maxima, hence obtaining a single value for the position each lane.



(a) Candidate Lane Points (b) Non Local Max Suppression

Figure 3.28: Resulting candidate lanes points (a) before and (b) after non local maxima suppression.

It is worth noting that in contrast with the method in [32] which focuses on the detection of the central lane, we added support for the detection of side lanes as well, provided they are visible in the bird eye view.

Finally, we implemented a way of calculating lane departure and keep track of lane changes which was also not mentioned in the original implementation. We assume that the car was centred on the lane when the lane centre is aligned with the centre of the region of interest. It follows that we are able to express the lane departure in terms of the difference between the x position of the ROI centre and the midpoint of the detected lane.

$$D = \frac{w}{2} - \frac{x_l + x_r}{2} \tag{3.7}$$

where w is the width of the ROI and x_l and x_r are respectively the

For example, in the frame shown the width of the projected ROI is equal to 140 pixels and the two identified lane boundaries are at $x = 49$ and $x = 85$ pixels. Therefore the lane departure is equal to:

$$D = 70 - \frac{49 + 85}{2} = +3 \tag{3.8}$$

We can express this number in percentage of half the lane width obtaining a lane departure of 8.3% to the right.

Upper ROI In the upper region of the region of interest we applied the proposed modified river-flow method to detect the remaining parts of the lanes [32].

To start with, the frame is thresholded with a dynamic value found analysing the pixel intensity histogram of the grey-scale ROI image. An example of pixel intensity histogram is shown in Figure 3.29.

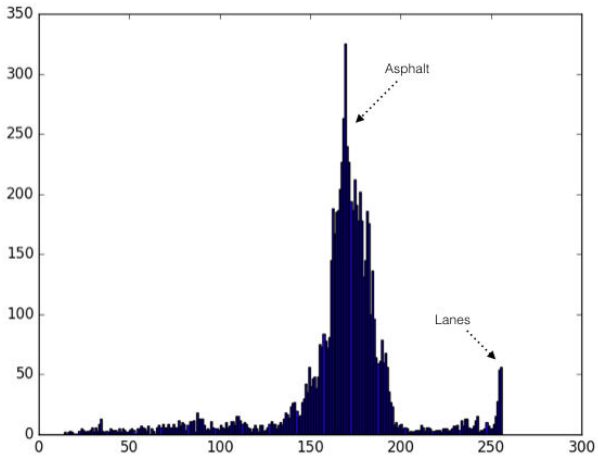


Figure 3.29: Pixel intensity histogram in the ROI selected for lane detection.

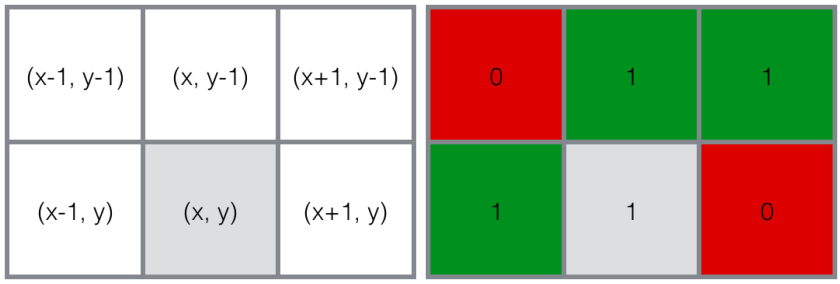
Clearly, in the example shown, the optimal threshold should be chosen in the middle of the two peaks in order to maximise the separation between them. In this case $T \approx 225$.

Figure 3.30: ROI after dynamic threshold applied.

Once the image has been thresholded (Figure 3.30) we can select a starting point for the river-flow method in the upper edge of the lower ROI. Such point is easily found as lanes in the lower ROI have already been identified.

The idea behind the river-flow approach is to *walk* inside a region of space with some selected properties (in this case high pixel intensity) in a pre-determined set of directions. In other words, if a pixel (x, y) satisfying the required conditions is currently selected, the condition to move forward is that a pixel satisfying the same conditions exists in a specially designed neighbourhood of the current pixel.

Figure 3.31 illustrates the chosen neighbourhood together with an example. The current pixel is always marked in light grey.



(a) Neighbourhood (b) Practical Example

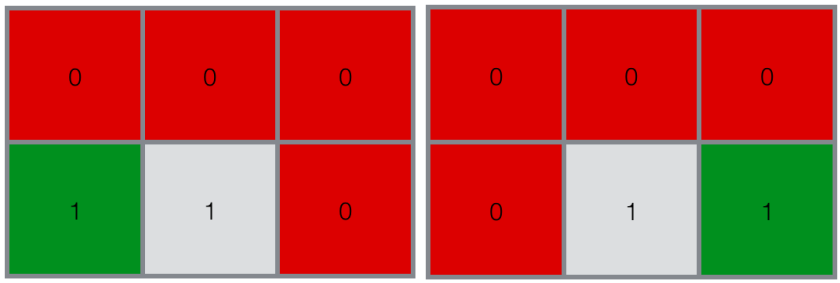
Figure 3.31: (a) Chosen neighbourhood for river-flow method; (b) practical example: green labelled pixels are all valid points for the forward movement of the algorithm.

Finally, it is important to define stopping conditions to the river-flow which have not been discussed in [32]. We defined two:

- The y coordinate goes below a specified value [2]
- The y coordinate does not change (i.e. decrease) in more than k iterations

where k is an arbitrary parameter that was empirically set to $k = 20$.

The first condition prevents the method from looking for lanes too above in the frame where they become much less likely to be found; the second condition prevents a dead-lock in which the method continuously bounces back and forth between horizontally adjacent pixels (Figure 3.32).



(a) (b)

Figure 3.32: Illustration of a dead-lock situation: e.g. (a) odd iterations; (b) even iterations.

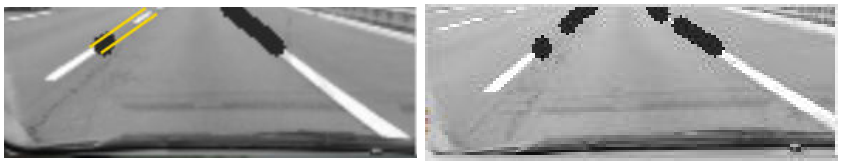
[2] Recall that the origin of an image is in the top-left corner

The resulting candidate lanes points are shown for the right lane boundary in Figure 3.33.

Figure 3.33: Candidate upper ROI lane points using the river-flow algorithm.

As it is shown in Figure 3.33, one problem is still to be solved. In fact, in order to be able to detect discontinuous lane boundaries we have to be able to jump from one segment to the following. We can achieve this by considering a search region as illustrated in Figure 3.34a. For convenience, the region is chosen to be parallel to the linear fit of the lane as calculated in the lower ROI. To reduce segment misses, we can make the search region as wide as appropriate, without forgetting that a wider search region yields an higher computational complexity.

Figure 3.34b shows the candidate left and right lane boundary points as a result of the river-flow method.



(a) Search Region (b) Final Result

Figure 3.34: Illustration of (a) the search region in the modified river-flow and (b) resulting candidate lane points.

Interpolation and Lane Equation

For increased robustness one could choose to interpolate the points using some pre-determined functions. In our system we initially chose a simple second order polynomial interpolation function (i.e. a parabola). This choice gave desirable results when fitting continuous lane boundaries but turned out to be not sufficiently tolerant to noise to fit discontinuous lanes (Figure 3.36a).

Therefore we modified the function as follows. Instead of fitting the points with a general equation:

$$x = ay^2 + by + c \tag{3.9}$$

we noticed that in the lower ROI the fitting curve should be tangent to the (approximately straight) lanes (Figure 3.35):

$$\left.\frac{dx}{dy}\right|_{y=h} = 2ah + b = k \tag{3.10}$$

where h is the height of the frame and k the slope of the line fitting the lane in the lower ROI[3].

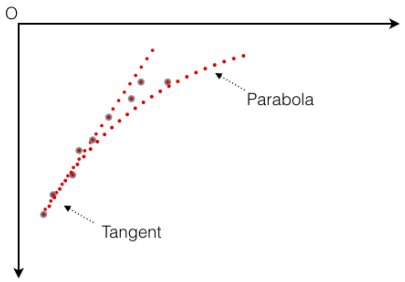


Figure 3.35: Example of polynomial interpolation of the left lane boundary.

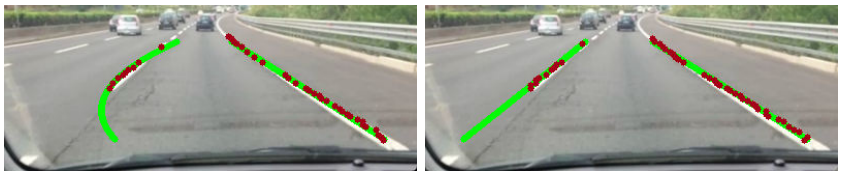
With such modification, the fitting equation becomes:

$$a(y^2 - 2hy_p) + c = x - y \left.\frac{dx}{dy}\right|_{y=h} \tag{3.11}$$

With enough (x, y) candidate lane points, the problem can be written in matrix form:

$$Y \begin{bmatrix} a \\ c \end{bmatrix} = x \tag{3.12}$$

and solved for a and c first and finally for b by substituting back into (3.10) (Figure 3.36b). Note that, for increased robustness, the RANSAC algorithm should be used.



(a) Standard Fitting (b) Improved Fitting

Figure 3.36: Illustration of (a) the search region in the modified river-flow and (b) resulting candidate lane points.

[3]Note that the origin is in the upper-left corner.

As a final remark, it is worth noting that in order to further aid the lane interpolation one could use a-priori information on the curvature of the road and the position of the car. The first can be calculated from maps and the second is not hard to get using a GPS receiver.

We did some initial research in this topic by writing code that interfaces to the Google Maps API and calculates the curvature along a given segment of the road (Figura 3.37). However, it was later clear that this topic would have been quite considerably outside the scope of the project and therefore we decided not to include any results in this report.

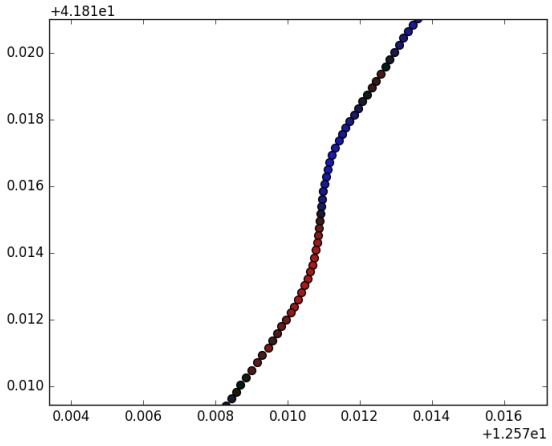
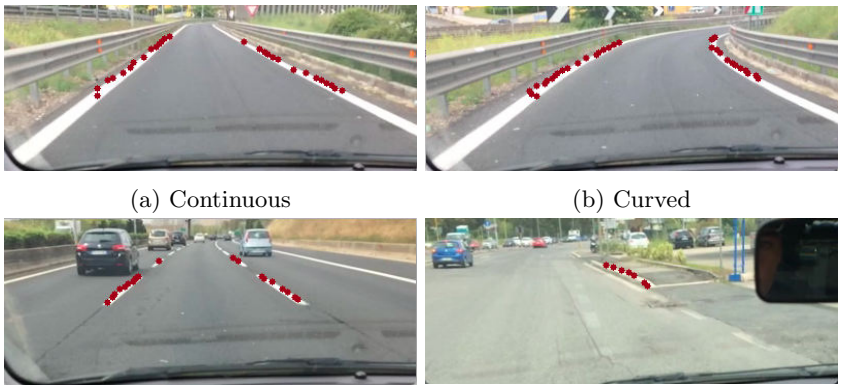


Figure 3.37: Calculation of the road curvature on a highway segment; lighter colour indicates a steeper curve; blue indicates a left turn and red a right turn.

Performance Assessment

We successfully tested the lane detector in a number of road types and conditions. In particular, in Figure 3.38, robustness to discontinuous lane boundaries, curved lanes and faded lane boundaries is shown.

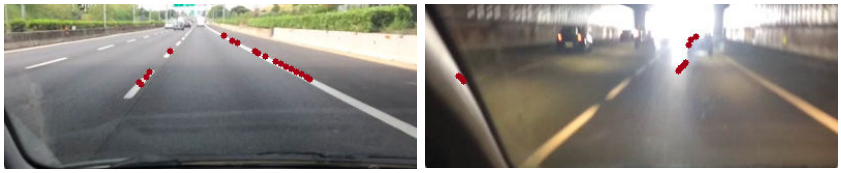


(a) Continuous (b) Curved

(c) Discontinuous (d) Faded

Figure 3.38: Examples of successful lane detection using the proposed algorithm.

Robustness to shadows can be obtained if the dynamic thresholded ROI is replaced with an edge detected ROI as shown in Figure 3.39a. However, since the local feature matching based head tracking method under-performs in low light conditions, such modification does not improve the performance in the example shown in Figure 3.39b.



(a) Shadows (b) Low Light Conditions

Figure 3.39: Performance of the algorithm under (a) shadows and (b) low light conditions. The poor performance in (b) is due to the loss of head track.

Among the challenges for a lane detector reported in [15] we find vehicle occlusion and adverse weather conditions too. However, our dataset does not include any driving session in bad weather; and in the current implementation we have not tackled the issue of vehicle occlusion: Figure 3.40 shows that the successful lane detection depends on the colour of the occluding vehicle.

(a) Dark Occluding Vehicle (b) Light Occluding Vehicle

Figure 3.40: Performance of the lane detector under vehicle occlusion.

A solution to the vehicle occlusion problem could be designed by combining the lane detector with the multi-class object detector (see 3.5.2) and avoid looking for lanes in regions of the image that contain occluding objects [32].

Key

Finally and most importantly, we show in Figure 3.41 that the successful detection of lane boundaries does not depend on the direction where the driver is looking. Clearly this would be impossible with a static selection of the region of interest. This result is therefore entirely due to our head tracking and dynamic ROI extraction methods.

(a) Straight (b) Down (c) Left

Figure 3.41: Examples of successful lane detection using the proposed algorithm. The successful detection does not depend on the orientation of the driver's head.

The analysis presented so far offers a qualitative proof of the very good performance of the lane detector. For completeness, we manually labelled 200 non consecutive frames of video and run the lane detector on them. In such experiment we measured a lane detection accuracy of 95% which is in line with the one of state of the art lane detectors that use a steady camera [1].

3.5.2 Multi-Class Object Classifier

To complete our system, in this subsection we present a review of state of the art multi class object classifiers.

Random Forest Classifier

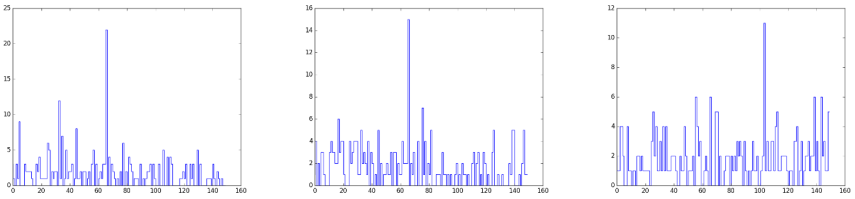
The random forest is a rather popular machine learning technique which is used extensively for multi-class object classification. Here we combined the RF classifier with a bag-of-words (BoW) model. The BoW treats image features as words of a dictionary and models every image as a histogram representing the number of occurrences of each of the features from the image itself in such dictionary.

The name bag-of-words comes from the first use of the method which was related to literature gender classification. However, in contrast with the original BoW model, the visual dictionary is not known before hand but rather it is calculated by clustering the entire set of features across all training images into a small number of cluster centres (usually tens or few hundreds).

Building the Codebook To build the codebook (or dictionary), features are extracted from training images. Features are then clustered with the K-means algorithm. Once the K-means have been calculated, for each image all of its features are associated with the respective closest cluster; once all features have been matched to a cluster, an histogram representation of the image can be built: each bin of the histogram represents a cluster and the height of the bin represents the number of occurrences of that cluster in the image, or rather the number of features in the image that have been matched to that cluster. [14]

RF Training Through the codebook, a significant dimensionality reduction is achieved. Now entire images are represented by a histogram of a few tens (or hundreds) of bins. Once all histograms have been calculated, they are fed to the random forest for training. This, depending on the dataset size and the chosen model parameters could take from a few minutes to several hours.

Figure 3.42 shows the histogram representations of three images: two negative samples and one positive sample from the CALTECH car dataset. It is interesting to note the first two images have a very similar distribution of clusters compared to the third image whose histogram looks quite different. It is the intra-class similarity and inter-class variability that allows the RF to discriminate between classes.



(a) Negative (Road) (b) Negative (Road) (c) Car

Figure 3.42: Histograms of (a)(b) two negative samples and (b) one positive sample from the CALTECH car dataset.

RF Testing Once the random forest is trained, it can be easily tested by passing a histogram from a test image to it. The histogram of a test image is obtained by using the same K-means clustering as obtained for the training set. For each tree in the forest, the testing data will reach a leaf node which a class distribution belongs to. To make a decision on the class, the outputs of the trees' ensemble are averaged and the class corresponding to the highest probability is retained. This passage is illustrated in Figure 3.43.

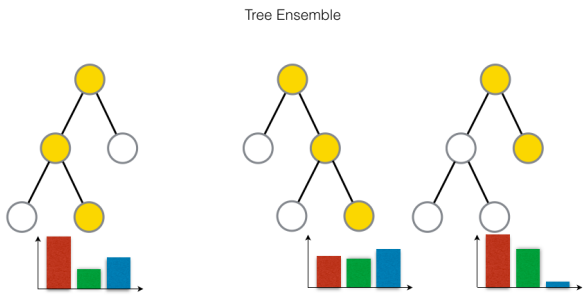


Figure 3.43: Illustration of random forest testing; note that decision trees are different in structure due to randomisation in training. Clearly, once the three class distributions are averaged, the red class would be predominant.

Results In supervised-learning algorithms, one of the most tedious jobs is to prepare a dataset, which in computer vision mostly involves labelling every training image with the correct class. This can take days of very repetitive work which is the reason why we decided to prove the concept by training and testing on a different dataset than ours. In particular, we used the CALTECH car dataset and the MIT pedestrian dataset.

(a) Car Sample

(b) Pedestrian Sample

(c) Negative Sample

Figure 3.44: Example images from (a)(c) the CALTECH car dataset and (b) the MIT pedestrian dataset.

We randomly split the dataset into training and test images and trained our model onto the training set. Then we built the K-means codebook and hence trained the random forest on the randomly selected training data. Finally, to evaluate the testing accuracy we ran the random forest on the half of the dataset that was not used for training.

The confusion matrix (Figure 3.45) shows that the RF performs very well on the dataset and it is particularly good at detecting pedestrians (class 2). Class 0 and class 1 are less discriminated. This is most probably due to the fact that cars are not well cropped out of the street as shown in Figure 3.44. In general, the testing performance is satisfactory and it demonstrates the ability of the RF to generalise classification the problem avoiding over-fitting of data [4].

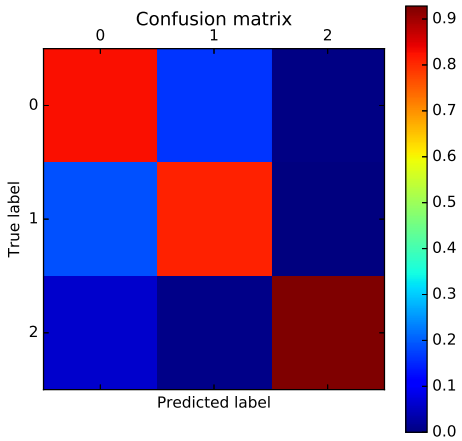


Figure 3.45: Confusion matrix of the random forest output. The classifier works best with pedestrians (class 2) and is less able to discriminate between cars (class 1) and negative samples (i.e. empty road, class 0).

System Integration In order to integrate the multi-class detector in the current system, it is possible to apply the random forest to every frame through a sliding window. Clearly, if this were done on a single core of a CPU, it would be impossible to get real-time performance. However, exploiting the inherently parallel structure of the forest, one could run the classifier on a large multi-core CPU or even on a GPU to get much more desirable performance.

YOLO Multi Object Detector

According to literature [22], to get an even better detection performance, one could use deep learning methods based on convolutional neural networks. The purpose of the next section therefore is to present one of such methods: You Only Look Once (YOLO). YOLO has been proven to be an extremely fast and accurate multi-class detector whose performance would definitely suite our requirements.

In contrast with most multi-class object detectors, YOLO does not use a sliding window. Furthermore, YOLO does not have a complex pipeline like most CNN for object detection and classification; rather it looks at the image altogether and simultaneously detects and classifies objects. YOLO uses a single neural network with 24 convolutional layers followed by 2 fully connected layers. The network is pre-trained for classification using low resolution images and then trained again for detection using higher resolution images [22].

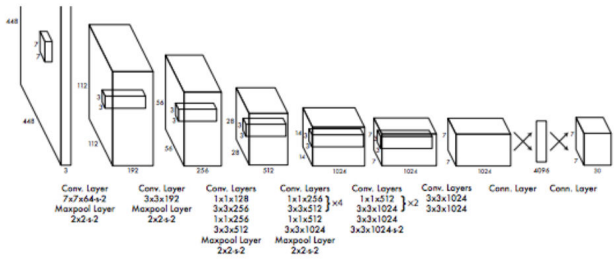


Figure 3.46: Diagram of YOLO CNN architecture.

The code for YOLO as well as many pre-trained models are available open source. It is often the case that the performance of pre-trained models varies significantly across different datasets. On the contrary, the ability of the YOLO network to generalise the problem of detection and classification makes it exhibit top performance on any test datasets (including art works) [22].

Performance Assessment

The significantly better performance according to literature as well as the availability of pre-trained models encouraged us to use YOLO as the default multi-class object detector in our system. In what follows, similarly to what we have done for the lane detector, we study the benefits of the head tracking and dynamic ROI selection on the performance of this algorithm.

In particular we start with two observations: first, from the YOLO neural network schematic diagram one can see that the algorithm is designed to work on 448×448 input images (Figure 3.46), which implies that a generic image must be first scaled and cropped in order to be compatible; secondly the NN architecture itself imposes some limitations on the density of detected objects which is the claimed reason why the algorithm does not perform exceptionally well on small objects, especially if they appear close to one another [22].

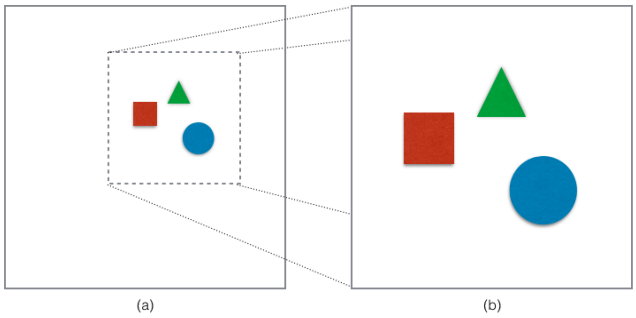
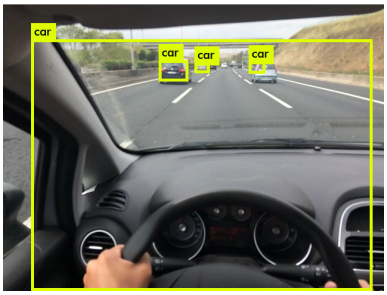


Figure 3.47: Illustration of the benefit of cropping and scaling up a region of interest in the input image. The size and the separation between the objects increase.

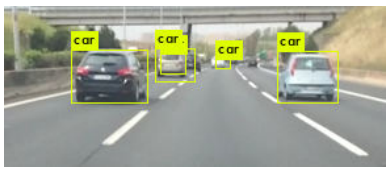
Key

It is not difficult to see then that by cropping those areas of the input image that are dense of features of interest we can increase the size as well as the separation of those features and hence improve the detection performance (Figure 3.47).

Figure 3.48 shows this concept developed onto a real example. It is particularly interesting to note that the detection performance improves by cropping out a ROI although no additional information is added about the detected features (i.e. their resolution remains the same): the improvement is only a result of the increased object size and of the better separation.



(a) Original Frame



(b) Cropped Frame

Figure 3.48: Example of better detection performance obtained by cropping a region of interest from the original frame.

Table 3.8 summarises the detection performance (with and without ROI extraction) of vehicles closer than $100m$ using the YOLO-tiny model. The test

was conducted on 200 non consecutive frames labelled with non-occluded cars closer than approximately $100m$[4]. Results are in line with the claimed YOLO-tiny model accuracy [23] with the performance being significantly better on the extracted ROI image.

Table 3.8: Comparison of car detection accuracy in the raw input image and in a cropped subregion of the input image.

	Full Frame	Extracted ROI
Detection Rate	61.76%	73.52%
False Positives Rate	13.0%	3.5%

3.6 User Interface

The ultimate goal of our system is to be able to generate virtual content based on the elaboration of the data discussed so far in order to complement the driver experience.

Thanks to the very accurate head tracker implemented in section 3.3, it is possible to display content in a number of different ways. Here we present a few.

Object Highlight Highlighting object with coloured boxes is the simplest method to display information to the driver. It is quite invasive and hence it is suitable for vital alarms such as collision avoidance.

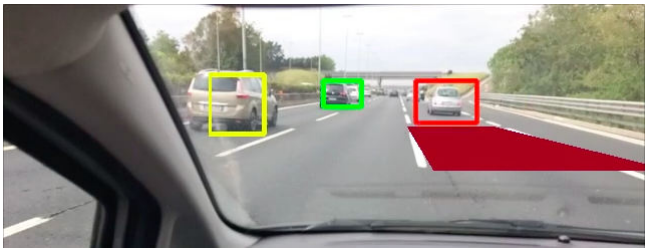


Figure 3.49: Example of object highlighting aimed at collision avoidance alerts.

Cockpit Aligned The second type of interface that one could design is related to the augmentation of reality inside the cockpit. This concerns for example the easier to read and less distractive display of car metrics such as the vehicle speed. In particular, head tracking can be used to pin information to some user

[4]The real distance between the camera and the surrounding vehicles was estimated manually with the aid of *satellite view* and *street view* on Google Maps

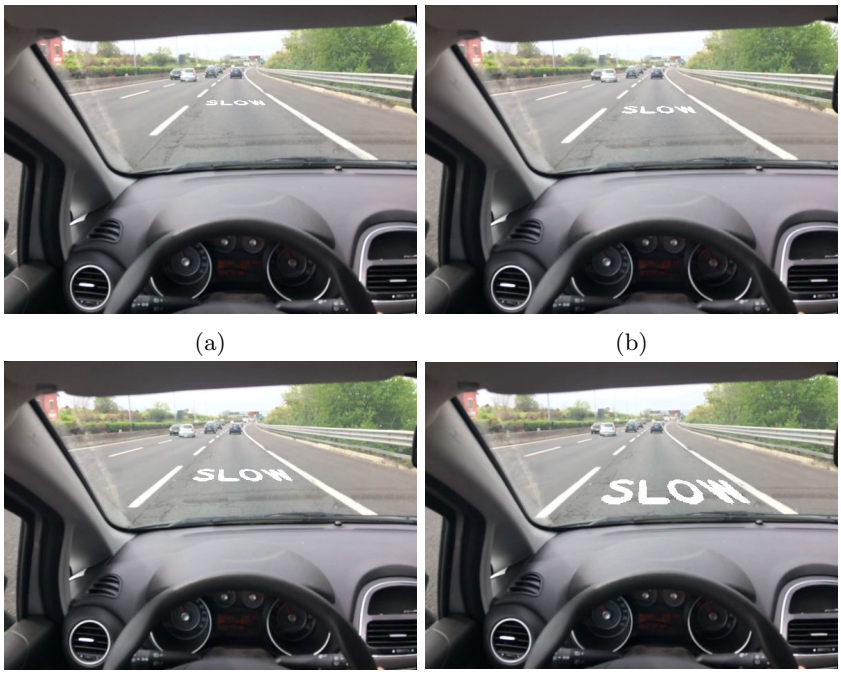
defined regions of the cockpit. Figure 3.50 shows the speed pinned on top of the steering wheel.

Figure 3.50: Virtual head-up display for speed indication. The speed is pinned on top of the steering wheel as a result of the head tracking.

Road Aligned Finally, the last and most interesting way of displaying warnings to the driver is by projecting it virtually onto the road level through a similar transform used in the lane detection algorithm (3.5.1). Such warnings may include speed limit, traffic information and lane departure warnings.

Figure 3.51: Example of road aligned information display. The information is displayed through a projective transform.

By combining such projection with the real speed of the car calculated using the GPS receiver one could make it dynamic as shown in Figure 3.52.



(a) (b)

(c) (d)

Figure 3.52: Sequence of 4 consecutive frames showing the dynamic *slow* alert. The alert moves according to the speed of the car obtained from the GPS receiver.

Chapter 4

System Implementation & Demonstration

So far we have evaluated the components of our system individually. In this section we would like to demonstrate the performance of our system as a whole both in terms of accuracy and computational requirements. In what follows we assume that the head is tracked through the local feature matching based method and surrounding cars are detected with the YOLO-tiny NN model. The data is from our own dataset.

4.1 Python Implementation

4.1.1 Libraries

Hard requirements for the system to run are:

- Python 2.7
- Numpy
- OpenCV 2
- Darkflow[1]

4.1.2 Components

Main

The main script is responsible for reading the input data files and to loop through the video frames applying all the developed processing stages. Every processing stage is implemented in a separate script in order to make the software modular and simple to maintain and debug.

[1] https://github.com/thtrieu/darkflow

The required modules are imported as shown:

```
from source import frame_manager
from source import file_manager
from source import gps_map_manager
from source import head_tracker
from source import projective_lane_detector
from source import multiclass_obj_detector
from source import alarm_generation
```

Then the initialisation of all modules follows. In these lines of code, all modules are pointed to their respective required input data sources (e.g. reference frame, input video, gps data, etc.).

Once the initialisation is complete, the main loop can start.

```
1  while (1):
2
3          clock = frame_data[frame_number][0]
4
5          #calculate position, road curvature and speed from GPS data
6          [...]
7
8          #read input frame
9          input_frame = frame_m.read_input_frame()
10         input_frame = frame_m.undistort(input_frame)
11
12         #head tracker
13         homography = ht.update(input_frame, clock, gyro_data)
14
15         #get ROI for lane detection
16         scale = 0.4
17         process_frame = frame_m.get_ROI(input_frame, scale, homography)
18
19         #lane detection
20         M, lane_geometry, upper_ROI_lanes, departure = ld.process_input_frame
               (process_frame, scale, road_curvature)
21
22         #get ROI for multiclass object detection
23         scale = 1
24         process_frame = frame_m.get_ROI(input_frame, scale, homography)
25
26         #detect objects
27         obj_det_result = obj_det.detect_objects(input_frame[y_min:y_max,
               x_min:x_max])
28
29         #user interface
30         a_g.process(input_frame, obj_det_result, lane_geometry,
               upper_ROI_lanes, speed_m_per_s, departure)
31
32         #write output video
33         [...]
34
35         frame_number+=1
36         counter += 1
```

The code listing shows the (simplified) code of the main loop. It is worth noting that the implementation closely follows the design: first the head tracking module is invoked (line 13); hence lane detection is performed onto a selected region of interest (lines 14-20); then the object detector is applied onto a slightly

different ROI (lines 22-27); finally all the extracted information is elaborated and displayed to the user (line 30).

Frame Manager

By combining the result of the attitude estimation with the off-line created labelling data, the region of interest for road feature extraction can be dynamically calculated. This is taken care by the `extract_ROI` function in the module `frame_manager`.

It is worth pointing out that because of the way the two detectors we included in our system (i.e. the lane detector and the multi-class object detector) have been designed, it is necessary to extract two slightly different ROIs. This is done by calling the `extract_ROI` twice as already stressed while describing the operation of the main loop.

File Manager

The `file_manager` module offers file I/O utilities that are used to input data that is necessary to the main processing. Examples of such may be the GPS and IMU data files.

GPS & Map Manager

The `gps_map_manager` module is responsible for the calculation of navigation meta-data. Some of this data such as the curvature of the road can be used to refine the accuracy of some system blocks; most of the information however is used in the user interface module to generate visual content for the driver.

Head Tracking

The `head_tracker` module implements the compensation for driver's head movements described in 3.3. The interface to the external world is through the function `update` which is responsible for the actual coordination of the IMU+vision tracking methods. The (simplified) code listing is shown:

```
1  def update(self, frame, clock, gyro_data):
2
3          g_image = self.get_gray_scale_image(frame)
4
5          self.homography = self.gyro(self.homography, clock, gyro_data)
6
7          self.homography = self.vision(g_image, self.homography)
8
9          return self.homography
```

As discussed at design stage, the gyroscope provides a rough estimate of the head pose that is then refined through the use of vision. The benefit of including the gyroscope data in the calculations is best appreciated in case of sudden and fast rotations of the head (and hence the camera).

Note In the vision based head tracker we use ORB[2] feature descriptors and matched features using the OpenCV function `BFmatch` (brute-force matcher) for best accuracy results. Using the same resolution as our dataset and implementing the modifications proposed in 3.3, the number of features to match turns out to be sufficiently small to use a brute-force matcher.

However, we should mention that in alternative scenarios, OpenCV also supports the Fast Library for Approximate Nearest Neighbours (FLANN) which is designed to guarantee good computational performance even with significantly larger numbers of features to match.

Lane Detector

Once the ROI is extracted, road features detection can be performed. The `detect_lanes` function in the `projective_lane_detector` module detects road lane boundaries in the lower and upper ROI as described in 3.5.1. Lane detection in the lower ROI is very robust and it is also used to calculate the position of the car within the lane.

No particular note needs to be made about the implementation of the lane detector as it closely follows the design illustrated in 3.5.1.

Multi-Class Object Detector

The `multiclass_obj_detector` module offers a very simple interface to the *darkflow* implementation of YOLO which supports Python and OpenCV.

```
1  from darkflow.net.build import TFNet
2  import cv2
3
4  class obj_det:
5
6          def detect_objects(self, image):
7
8                  return self.tfnet.return_predict(image)
9
10         def __init__(self):
11
12                 path = "./data/"
13                 options = {"model": path+"yolo/tiny-yolo-voc.cfg", "load":
                       path+"yolo/tiny-yolo-voc.weights", "threshold": 0.1, "
                       gpu": 0.0}
14
15                 self.tfnet = TFNet(options)
```

In the `__init__` function, the neural network is loaded with the required `options`. The actual NN testing happens in the function `detect_objects` which takes an input image and returns a json file with coordinates, class and confidence of the detected objects.

[2]In development, we tested SIFT, SURF and ORB features, which all yielded very similar head tracking performance. In the end ORB was chosen for its most desirable computational requirements [28].

User Interface

The `alarm_generation` module collects all the detected features of the scene and extrapolates useful information to support (and hopefully simplify) the driving experience.

The understanding of a road scene is out of the scope of our research and much work nowadays is being done in this area using very advanced machine learning techniques.

Nevertheless, to prove the concept of what could be developed on top of the system we are presenting, we implemented a few alarm triggers which include vehicle collision avoidance, lane departure and over-speed alerts.

4.2 Demo

The system proposed opens up endless possibilities to augment the reality of the driver. In this section we wish to demonstrate a few ways to combine the data extracted in previous sections to complement the driving experience and generate warnings when necessary.

In particular let us consider the highway setting illustrated in Figure 4.2f. The car is moving onto a multi lane highway surrounded by other vehicles. To demonstrate the capabilities of our system we would like to produce collision alerts as well as to remind the driver of the presence of side vehicles when switching lane.

Figure 4.1 shows that our system is able to reliably track the position of the car onto the lane. Furthermore sudden changes between positive and negative lane departure measures correspond to lane switches.

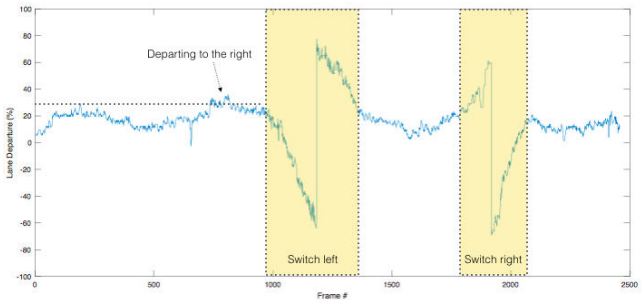
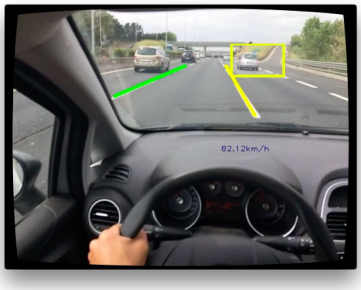


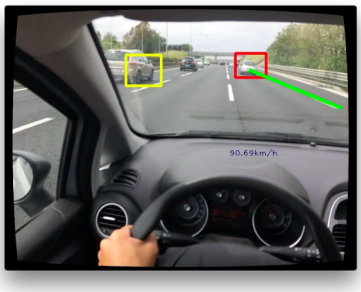
Figure 4.1: Example of lane departure estimate.

Figure 4.2 shows sample output frames of our system. Note that vehicles are colour coded according to their proximity and hence their risk of collision (Figure 4.2f). However, if the driver is keeping the lane correctly, he is not notified of side vehicles to avoid distraction. Lastly, the lane boundary colours turn from green to yellow if the lane is being departed.

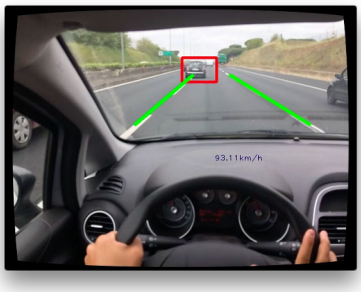
(a) Lane Departure Alert



(b) Lane Departure Alert

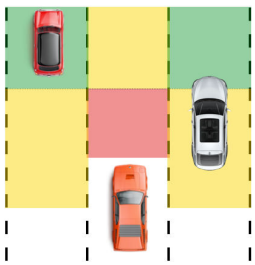


(c) Lane Switch Alert



(d) Collision Alert

(e) Over-speed Alert



(f) Colour Coding

Figure 4.2: Example of vehicle .

Note that with very little additional effort one could distinguish between continuous and discontinuous lane boundaries and hence extend the types of

alerts the lane departure system can generate.

4.3 Code Profiling

In this final part we briefly assess the computational performance of the system proposed. In particular we will show that even on a device with modest processing capabilities, our software is able to run in real time.

The code has been timed using the standard Python function `time()` on a MacBook Air 2012 with an Intel Core i5. This function measures the effective time in seconds; it follows that the performance of our system will be underestimated as other processes will be running on the CPU simultaneously. Nevertheless the experiment we describe should give a reasonable overview of the system computational requirements.

4.3.1 Head Tracking

The first block of our system is the attitude determination stage. If attitude estimation is performed with the gyroscope-assisted local feature matching method, then the average processing time is $\approx 0.056s$, distributed as shown in Figure 4.3.

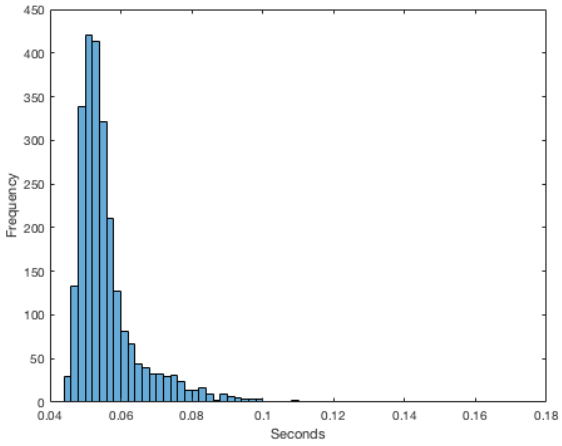


Figure 4.3: Timing of head tracking over 1200 frames.

From the distribution it can be estimated that to run the attitude determination block in real time dropping less than 10% of frames we should require $\approx 0.065s$.

4.3.2 Lane Detection

We can perform a similar analysis on the lane detector. In particular our results showed that the average processing time needed by the lane detection stage is $\approx 0.0316s$, distributed as shown in Figure 4.4. Note that this time the distribution has a higher variance: this is expected as, depending on the quality of the lane boundaries in the image, the detection may take significantly more (or less) than the average.

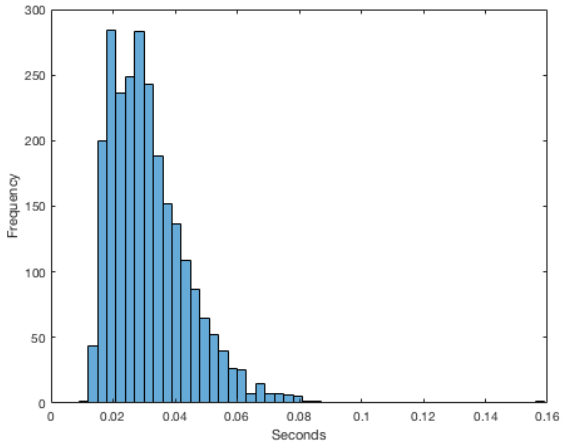


Figure 4.4: Timing of lane detection over 1200 frames.

Once again, from the distribution we can estimate the time that we have to allow for this processing stage in order to drop less than 10% of frames, obtaining $\approx 0.049s$.

4.3.3 Multi-Class Object Classifier

The YOLO multi-class object detector is one of the very few real time detectors that have been developed. In particular, its faster version, known as YOLO *tiny*, is able to run at around $200fps$ even on moderately sized GPUs [25]. YOLO *tiny*, which is slightly less accurate than the more complex YOLO, was designed to prove reliable multi-class object detection at high frame rates.

The only requirement to achieve the quoted performance is a GPU with at least 611MB of memory [24], a requirement which nowadays even mobile devices are very likely to meet.

4.3.4 UI

We omitted the computational requirements of the user interface in our experiment as results would be highly platform dependent. Moreover, OpenCV is particularly slow in rendering virtual content which instead would be highly optimised on AR libraries.

4.3.5 Summary

Assuming independence of the random processes behind the timing of the various processing stages we can claim that our system is able to run in $\approx 0.119s$ (or $8.4fps$) obtained as:

$$T = 0.065s + 0.049s + \frac{1}{200Hz} = 0.119s \tag{4.1}$$

excluding the computation required to render the user interface.

Clearly $8.4fps$ is still not sufficient to be able to render content smoothly in a AR device. Nevertheless we should not forget that the code we used for testing is executed sequentially as no optimisation is in place.

As a final remark, we should note that repeating the same analysis using the correlation based head tracker, which is less accurate but significantly faster, we could achieve real time operation without further optimising the code: $20.4fps$.

Chapter 5

Comments & Future Work

The project as it was formulated at the beginning of the academic year was rather open-ended and offered a large number of connected research topics to investigate. In order to make it fit in the given time constraints, we therefore selected the subset of such topics that constituted the core of this report.

The purpose of this chapter is to provide a qualitative evaluation of the project results, investigate opportunities of further development and reason about potential applications of the technology.

5.1 Comments

Overall, we can regard the project as successful as we were able to develop a software for a standalone AR device which, according to requirements, should have provided driver assistance while not relying on any of the vehicle's systems.

An important achievement has been the implementation of a novel method for head tracking which allowed us to make use of computer vision methods that made strict assumptions on the mounting of the camera system into an unbonded camera setting.

Unfortunately a pre-labelled dataset with a moving camera inside the car cockpit could not be found. Therefore we recorded our own set of data (published alongside this report) and, as much as we possibly could, we tried to provide a quantitative analysis of our results, although this implied labelling our dataset manually which took quite a significant time.

Finally, in some parts of the system and especially in the head tracker we made a few assumptions to slightly simplify the problem. It would be interesting, in future development of the project, to try and relax those assumptions to make

the device even more versatile and extend the AR experience beyond the driver to the other passengers of the vehicle.

5.2 Further Development Opportunities

Due to the limited amount of time and resources, there was a number of topics of interest to the project that we decided not to explore. Nevertheless, in what follows we would like to propose a few ideas that we believe should be object of future research in this AR system.

5.2.1 Pose Determination Using PnP

To start with, we wish to illustrate the Projective-n-Point (PnP) method as a way to relax the assumptions made on the Caresian position of the driver's head. In particular, this method offers a solution to the determination of the full 6 DOFs camera pose in a *known* 3D environment.

Among the benefits of tracking the driver's head with the PnP algorithm, we highlighted the potential extension of the AR experience, beyond the driver, to other passengers in the vehicle. Clearly passengers would not benefit from the same type of information (e.g. collision alert); rather they would be at the centre of a new AR based in-cockpit entertainment system.

Theory

PnP uses correspondences between the 3D world and the image plane and exploits simple geometry to estimate the camera pose. The simplest of PnP algorithms is known as P3P and uses only three points to estimate the camera pose. Figure 5.1 gives an illustration of P3P.

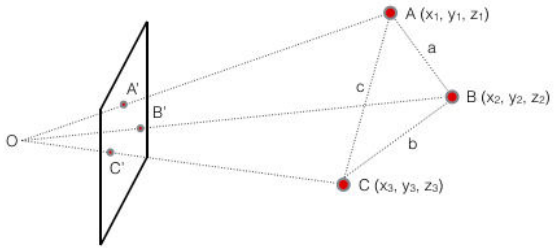


Figure 5.1: Illustration of the P3P solution.

From Figure 5.1, we can write a triple of equations whose solution is the solution to the P3P problem:

$$\begin{cases} a = (AO)^2 + (BO)^2 + 2(AO)(BO)\cos(\angle AOB) \\ b = (CO)^2 + (BO)^2 + 2(CO)(BO)\cos(\angle COB) \\ c = (CO)^2 + (AO)^2 + 2(CO)(AO)\cos(\angle COA) \end{cases} \tag{5.1}$$

Note that the equations follow from the cosine theorem:

$$c = a^2 + b^2 + 2ab\cos(\alpha) \tag{5.2}$$

where a, b and c are sides of a triangle and α is the angle $\angle ab$ (opposite to c).

In (5.1) the unknowns are $X = AO$, $Y = BO$ and $Z = CO$. a, b and c are known from the 3D map of the environment and the angles $\alpha = \angle COB$, $\beta = \angle COA$ and $\gamma = \angle AOB$ can be obtained from the image, provided that the internal camera parameters are known (focal length, CCD size, etc.). Discarding a solution that would put the 3D points behind the image plane, the knowledge of X, Y and Z fully determines the 6 DOFs pose of the camera.

If more points than three are available, the estimate can improve. Methods exist that are able to solve the PnP problem in time that is linear in the number of points. One example is the EPnP algorithm [18].

Practical System Integration

The PnP method is significantly more complex than the local feature matching proposed to solve the head tracking problem. Not only does it require more computation to solve for the device pose but it also needs quite a significant off-line effort to build a 3D map of the environment.

Off-line One possibility to build such a map is to use a radar or laser depth sensor paired with an IMU and a camera. The depth sensor and the IMU are used to record the 3D perception of the environment while the camera is used to extract local features. The 3D map is then labelled with such features.

On-line The on-line system is much more similar to the one we have already described in the local feature matching based head tracker. Local features are extracted from the input frames; such features are then matched with those of the 3D map hence building the image plane to 3D world list of correspondences which is necessary for PnP; finally, PnP is applied to determine the 6 DOFs pose of the camera.

5.2.2 Innovative User Interface

So far we have only covered one aspect of the user interface, that is the device to user side in order to display AR content. We would like to stress that an innovative interface could be built to let the driver communicate with the device too.

As an example, a simple extension of the existing code could let the software recognise driver's hand gestures which would open endless possibilities: controlling the radio, setting the cockpit temperature, picking up calls; all without getting distracted from the road.

5.3 Applications

The applications that such device could have are also very broad. Here we mention a few.

5.3.1 Research

Clearly the first application that comes to mind given the amount of driving data that is generated during every run of the application is research. One could study the behaviour of the user, understand its driving style, how he reacts to critical situations.

As a result, more tailored alerts could be generated to the user or one may even use the resulting data to train a computer to adopt the driving style of a human.

5.3.2 Driving School

Another interesting application is the analysis and assessment of driving style which would find an application in driving schools or may simply encourage safety in more experienced drivers.

Checking mirrors before changing lanes, keeping the correct distance from the leading vehicle are only few of the things that can already be checked for under the current implementation.

5.3.3 Marketing

Finally, AR could be used to generate non-invasive advertisement tailored to the user instead of using static real panels on the side of the road like it is done today.

Part II

Extras

Chapter 6

Data Collection

6.1 Introduction

Large part of the data necessary for the success of the project was not available publicly. In fact, even though a lot of material exists related to the topic of road features detection in form of source code and raw data for analysis (e.g. pictures for classifiers' training, on-road video shootings, etc.), very little if anything could be found to aid the research on head mounted AR devices for driving applications.

As a result, as discussed in previous sections of this report, we were forced to collect our own set of data. The information we were after was not limited to video shootings and therefore a normal camera could not be used. Rather we developed an iPhone application in order to record video frames and sensor data (e.g. IMU, GPS, etc.) in a synchronised way.

6.2 iPhone Application

The iPhone application has been designed and implemented uniquely for the construction of a database for the project. As a result it is a very simple single view application with very limited user control: the only button allows the user to start and stop the data recording.

The collected data can be accessed and downloaded through the Xcode Device Manager found in "Window → Devices". Once in the device manager, it is enough to select the (connected via USB) device in the left panel and then download the application *container*. The downloaded container can be opened in Finder to find the data at the Path "Local → Documents".

The iPhone application was designed to collect the following pieces of information from the device's sensors:

- Video Frames (Camera)
- Coordinates and Course (GPS Receiver)
- Roll, Pitch, Yaw (Accelerometer, Magnetometer, Gyroscope)

Data from the multitude of sensors is synchronised using standard time stamps (time in milliseconds since January 1st 1970).

In what follows we illustrate the main hardware settings as specified in the application.

6.2.1 Camera

Set-up The iPhone camera was set up according to the following code:

```
1  session = [[AVCaptureSession alloc] init];
2
3  //Set video preset
4  if ([session canSetSessionPreset:AVCaptureSessionPreset640x480]) {
5          session.sessionPreset = AVCaptureSessionPreset640x480;
6  }
7
8
9  //Configure device
10 AVCaptureDevice *device = [AVCaptureDevice defaultDeviceWithDeviceType:
       AVCaptureDeviceTypeBuiltInWideAngleCamera mediaType:AVMediaTypeVideo
       position:AVCaptureDevicePositionBack];
11
12 if ([device isFocusModeSupported:AVCaptureFocusModeLocked]) {
13         NSError *error = nil;
14         if ([device lockForConfiguration:&error]) {
15                 device.focusMode = AVCaptureFocusModeLocked;
16
17                 [device unlockForConfiguration];
18         }
19 }
20
21 //Add input
22 NSError *error;
23 AVCaptureDeviceInput *captureDeviceInput = [AVCaptureDeviceInput
       deviceInputWithDevice:device error:&error];
24 if ([session canAddInput:captureDeviceInput]) {
25         [session addInput:captureDeviceInput];
26 }
27
28
29 //Add output
30 dataOutput = [[AVCaptureVideoDataOutput alloc] init];
31 videoDataOutputQueue = dispatch_queue_create("VideoDataOutputQueue",
       DISPATCH_QUEUE_SERIAL);
32 [dataOutput setSampleBufferDelegate:self queue:videoDataOutputQueue];
33 dataOutput.videoSettings = @{(id)kCVPixelBufferPixelFormatTypeKey:@(
       kCVPixelFormatType_32BGRA)};
34 if ([session canAddOutput:dataOutput]) {
35         [session addOutput:dataOutput];
36 }
37
38 [session startRunning];
```

Remarks There a few details worth noting in the camera set up code. Firstly the camera is manually set to focus on the far fields rather than on close objects (lines 12-19). Since both the camera cockpit and the road are in the field of view of the camera, this prevents the camera to adjust the focus on close and irrelevant items in the cockpit while leaving the road blurred.

Secondly, instead of writing asynchronously to a video file, the camera is set up to notify the application each time a frame is collected (lines 30-36). This way each frame can be associated to a time stamp and hence synchronised with other sensor data.

6.2.2 Position & Motion Sensors

GPS

Set-up The position sensor is the GPS. The configuration code is listed below:

```
1  // Start location services to get the true heading.
2  self.locManager.distanceFilter = 0.01;
3  self.locManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation;
4
5  if ([self.locManager respondsToSelector:@selector(
       requestWhenInUseAuthorization)]) {
6  [self.locManager requestWhenInUseAuthorization];
7  }
8
9  [self.locManager startUpdatingLocation];
```

Remarks The GPS receiver set up is very simple. The only parameter we specify is the desired accuracy which we set to the maximum available (line 3). When a new GPS reading is available, it gets associated to a time-stamp and written to the `gps.csv` file.

IMU

Set-up The accelerometer, gyroscope and magnetometer in the IMU can be accessed separately. The set up code is as follows:

```
1  #define PERIOD  0.01 //s
2
3  [...]
4
5  _manager.gyroUpdateInterval = PERIOD;
6
7  [_manager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
8  withHandler:^(CMGyroData *gyroData, NSError *error) {
9  [self outputRotationData:gyroData.rotationRate];
10 }];
11
12 _manager.accelerometerUpdateInterval = PERIOD;
13
14 [_manager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
       withHandler:^(CMAccelerometerData *accelerometer_data, NSError *error){
15 [self outputAccelerationData:accelerometer_data];
16 }];
17
```

```
18  _manager.magnetometerUpdateInterval = PERIOD;
19
20  [_manager startMagnetometerUpdatesToQueue:[NSOperationQueue currentQueue]
        withHandler:^(CMMagnetometerData *magnetometer_data, NSError *error){
21  [self outputMagnetometerData:magnetometer_data];
22
23  }];
```

Remarks The configuration of the IMU is as simple as the one of the GPS receiver. Once again, we only have to specify one parameter, in this case the update rate which we set to $0.01s$ or $100Hz$ (line 1). Furthermore, exactly in the same way as for the camera and the GPS receiver, when new data is available, a time-stamp is generated and everything is saved to a output file.

6.2.3 Log-Files Management

Once all data is collected, it gets saved onto log files. There is one for each sensor including one for the camera which maps frame numbers into time-stamps for synchronisation purposes.

6.2.4 Example

Figure 6.1 illustrates an example of video frame together with sensor data from the one longest data collection session: a 22.5km drive around Rome. The itinerary is also shown in Figure 6.2.

(a) Frame

Timestamp	Coordinates (rad)	Altitude	Course	GPS Accuracy
1482737422.342711,	0.73127624, 0.21949006,	31.86257553,	283.71093750,	5.00000000, 3.00000000
1482737424.592530,	0.73127625, 0.21949014,	31.50954819,	283.71093750,	5.00000000, 3.00000000
1482737424.672511,	0.73127626, 0.21949020,	33.50405502,	283.71093750,	5.00000000, 4.00000000
1482737452.116373,	0.73127673, 0.21949006,	31.29311752,	337.50000000,	5.00000000, 3.00000000
1482737453.135159,	0.73127720, 0.21948967,	31.26009750,	333.98437500,	5.00000000, 3.00000000
1482737454.105863,	0.73127759, 0.21948942,	31.22677231,	333.98437500,	5.00000000, 3.00000000
1482737455.082463,	0.73128045, 0.21948820,	31.39522934,	349.45312500,	5.00000000, 3.00000000
1482737456.277378,	0.73128167, 0.21948819,	31.37026596,	354.02343750,	5.00000000, 3.00000000

(b) Sensor data

Figure 6.1: Example of data collection session. Note that the camera is mounted on the passenger seat: the data belongs to the first version of the data set.

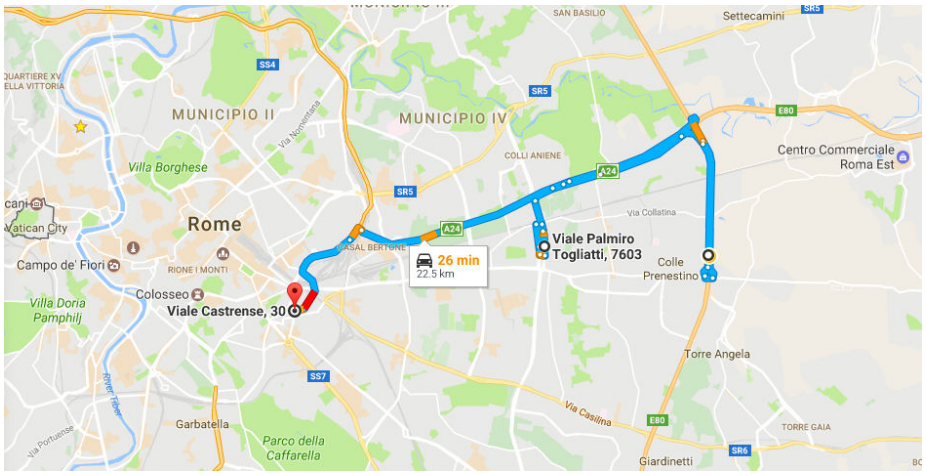


Figure 6.2: Itinerary of a 22.5km long data collection session.

6.3 Camera Calibration

The camera has been calibrated to account for radial distortion which is responsible for artefacts in the image corners. This is particularly important for the head tracker since corners are sometimes dense of features that are useful for the rigid transformation estimation.

Using the MATLAB *cameraCalibrator* app, a slight negative radial distortion was found (Figure 6.3c) and corrected with the radial coefficients:

$$\begin{aligned} k_1 &= 0.1919 \\ k_2 &= -0.2710 \\ k_3 &= 0.8019 \end{aligned} \tag{6.1}$$

An example of original and undistorted input frame is shown in Figure 6.3.

(a) (b) (c)

Figure 6.3: Camera calibration.

Bibliography

[1] Mohamed Aly. Real time detection of lane markers in urban streets. *2008 IEEE Intelligent Vehicles Symposium*, 2008. doi: 10.1109/ivs.2008.4621152. URL `http://www.vision.caltech.edu/malaa/publications/aly08realtime.pdf`.

[2] J. Joseph Antony and M. Suchetha. Vision based vehicle detection: A literature review. *International Journal of Applied Engineering Research*, 11(0973-4562):3128, 2016.

[3] Mohamad Nazree Dol Bahar and Xandri Farr. Modular cmos horizon sensor for small satellite attitude determination and control subsystem. *20th Annual AIAA/USU Conference on Small Satellites*, 2006. URL `http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1592&context=smallsat`.

[4] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using random forests and ferns. *2007 IEEE 11th International Conference on Computer Vision*, 2007. doi: 10.1109/iccv.2007.4409066. URL `http://www.cs.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf`.

[5] European Commission. *Advanced driver assistance systems*. URL `https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/ersosynthesis2016-adas15_en.pdf`.

[6] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pannern Analysis and Machine Intelligence*, 34(4):743–743 – 761, 2012. doi: 10.1109/TPAMI.2011.155.

[7] Amartansh Dubey and K. M. Bhurchandi. Robust and real time detection of curvy lanes (curves) having desired slopes for driving assistance and autonomous vehicles. URL `https://arxiv.org/pdf/1501.03124.pdf`.

[8] Steven J. Dumble and Peter W. Gibbens. Horizon profile detection for attitude determination. *Journal of Intelligent & Robotic Systems*, 68(1573-0409):339–339–357, 2012.

[9] Kamarul Ghazali, Rui Xiao, and Jie Ma. Road lane detection using h-maxima and improved hough transform. *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation*, 2012. doi: 10.1109/cimsim.2012.31.

[10] A. Giachetti, M. Campani, and V. Torre. The use of optical flow for road navigation. *IEEE Trans. Robotics and Automation*, 11(1):34–34–48, 1998.

[11] U.S. Government. Gps accuracy. URL `http://www.gps.gov/systems/gps/performance/accuracy/`.

[12] Wang Jingyu and Duan Jianmin. Lane detection algorithm using vanishing point. *2013 International Conference on Machine Learning and Cybernetics*, 2013. doi: 10.1109/icmlc.2013.6890384.

[13] Heechul Jung, Junggon Min, and Junmo Kim. An efficient lane detection algorithm for lane departure detection. *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013. doi: 10.1109/ivs.2013.6629593.

[14] Tae/Kyun Kim. Visual'codebook. URL `http://mi.eng.cam.ac.uk/~cipolla/lectures/PartIB/old/IB-visualcodebook.pdf`.

[15] Ammu M Kumar and Philomina Simon. Review of lane detection and tracking algorithms in advanced driver assistance system. *International Journal of Computer Science and Information Technology*, 7(4):6578, 2015. doi: 10.5121/ijcsit.2015.7406.

[16] David C. Lee. Boosted classifier for car detection. *ECE Department Carnegie Mellon University U.S.A.* URL `http://www.cs.cmu.edu/~dclee/car_boosted.pdf`.

[17] Yu-Chi Leng and Chieh-Li Chen. Vision-based lane departure detection system in urban traffic scenes. *2010 11th International Conference on Control Automation Robotics & Vision*, 2010. doi: 10.1109/icarcv.2010.5707817.

[18] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *Int J Comput Vis*. URL `http://cvlabwww.epfl.ch/~lepetit/papers/lepetit_ijcv08.pdf`.

[19] H. Mallot, H. Bulthoff, J. Little, and S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*, 64(3):177–177–185, 1991.

[20] ODG. R-7 smartglasses. URL `https://shop.osterhoutgroup.com/products/r-7-glasses-system`.

[21] Talat Ozyagcilar. Calibrating an ecompass in the presence of hard- and soft-iron interference. Technical Report AN4246, Freescale Semiconductor, 2015.

[22] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL `http://arxiv.org/abs/1506.02640`.

[23] Joseph Chet Redmon. Imagenet classification, 2016. URL `https://pjreddie.com/darknet/imagenet/#reference`.

[24] Joseph Chet Redmon. Yolo: Real-time object detection, 2016. URL `https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection`.

[25] Joseph Chet Redmon. Yolo: Real-time object detection, 2016. URL `https://pjreddie.com/darknet/yolo/`.

[26] Edgar Riba Pi. *Implementation of a 3D pose estimation algorithm.* PhD thesis, 2015. URL `http://upcommons.upc.edu/bitstream/handle/2117/77555/Master%20Thesis%20Riba%20Pi.pdf?sequence=1`.

[27] Raffaele Rinaldesi. *AUTOSTRADA A1: MILANO NAPOLI AMPLIAMENTO ALLA TERZA CORSIA.* URL `http://www.va.minambiente.it/File/Documento/20446`.

[28] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. URL `http://www.willowgarage.com/sites/default/files/orb_final.pdf`.

[29] Davi Antnio Dos Santos, Raphael Ballet, der Alves De Moura Moura, and Luiz Carlos Sandoval Ges. Visual-inertial request for attitude determination of multirotor aerial vehicles. *Anais do IX Congresso Nacional de Engenharia Mecnica*, 2015. doi: 10.20906/cps/con-2016-0648.

[30] SAE International Global Ground Vehicle Standards. Automated driving: Levels of driving automation are defined in new sae international standard j3016. URL `https://www.sae.org/misc/pdfs/automated_driving.pdf`.

[31] Z. Sun, G. Bebis, and R. Miller. On-road vehicle detection: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(5):694–694–711, 2006.

[32] Huachun Tan, Yang Zhou, Yong Zhu, Danya Yao, and Keqiang Li. A novel curve lane detection based on improved river flow and ransac. *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014. doi: 10.1109/itsc.2014.6957679.

[33] Z. Tao, Ph Bonnifait, V. Fremont, and J. Ibanez-Guzman. Mapping and localization using gps, lane markings and proprioceptive sensors. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013. doi: 10.1109/iros.2013.6696383.

[34] Paul Viola and Michael J. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57:137154, 2004.

[35] Yue Wang, Dinggang Shen, and Eam Khwang Teoh. Lane detection using spline model. *Pattern Recognition Letters*, 21(8):677689, 2000. doi: 10.1016/s0167-8655(00)00021-0.

[36] WindowsCentral. These are the full hardware specifications of the microsoft hololens. URL `https://www.windowscentral.com/hololens-hardware-specs`.

[37] Oliver J. Woodman. An introduction to inertial navigation. Technical Report 696, University of Cambridge, 2007. URL `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf`.

Appendix A

Code

Code is available at https://github.com/alino95/final_year_project. Note that the repository is private. For access email: alessandro.versini13@imperial.ac.uk.

Appendix B

Dataset

The dataset is publicly available at https://drive.google.com/drive/folders/0BxH-wKuCSdWpQnZXcHBHZURONzg?usp=sharing. For any issues, please email alessandro.versini13@imperial.ac.uk.

Appendix C

Project Proposal

Industrial Project Proposal

Title:

Augmented Reality Based Assistance for Driving Applications

Introduction

The purpose of this industry led final year project is to investigate the application of augmented reality techniques in the automotive industry. The project will focus on several aspects of the driving experience, from collision prevention to navigation. The novelty of the project will be in the use of the knowledge of driver's head and eyes position to present information in a way that is most tailored to the driver's attitude.

Project Plan

The base for the project will be a lane detection system that will be modified to add a number of information that will enrich the driving experience (discussed below). Several lane detection methods currently exist and are well compared and explained by Kumar and Simon (2015)[1]. The first part of the project will hence consist in a review of literature to determine the optimal lane detector for urban environments. An implementation of such detector will immediately follow.

Following the identification and successful implementation of the lane detector, the system will be modified to identify features surrounding the car (e.g. pedestrians, cars, etc.) and present tailored collision alerts and/or navigation data using information on eye and head position. The student will finally evaluate the feasibility of the installation of such a system in a pair of augmented reality glasses.

Project Supervision

The project has been discussed with prof. Yiannis Demiris and will make use of equipment already available in the Personal Robotics Lab: HTC Vive with fast eye detection, Driving Simulator with head and eye position detection, programmable robots to simulate obstacles and pedestrians. The student will develop the necessary software and tailor the hardware available in the lab to suit the required experiments.

Project Timeline

Checkpoint	Date
Review of existing lane detectors	Mid October
Implementation of lane detector	Early November
Review of existing object identification methods	Christmas Holiday
Study of head and eye tracking systems	January
Implementation of object identification	February - March
Implementation of eye and head trackers	March - April
Full system implementation	May - June

References

[1] Kumar , A.M, Simon, P. Review of Lane Detection and Tracking Algorithms in Advanced Driver Assistance System. . 2015;