eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO

INGENIARITZA ESKOLA

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

METHODOLOGY FOR CAR LOCATION AND MOTION DETERMINATION USING CAMERA SENSORS AND IMAGE SEGMENTATION AND CLASSIFICATION

Document 1 – Memory

Document 2 – Anex I: Position estimator code

Document 3 – Anex II: Road classifier code

Estudiante	*Ander Ramos Moreno*
Director	*Mikel Díez Sánchez*
Co-Director	*Asier Zubizarreta Picó*
Departamento	*Automática y control*
Curso académico	*2019-2020*

Bilbao, 12, Septiembre, 2020

ABSTRACT

In this document a study of the different methodologies to obtain the position of an autonomous vehicle by using camera images is carried out. The objective of the project is not only to calculate the position change of the car from one frame to the following one, but also to study the different systems to relocate the car in a map by detecting characteristic features of the environment in the image, allowing to correct the accumulated deviation. Among the different options for doing this relocalization, it is developed a road segmentation and classification system, as it is considered to be the most interesting option for the deviation correction problem.

RESUMEN

En este documento se lleva a cabo un estudio de las diferentes metodologías para obtener la posición de un vehículo autónomo usando imágenes obtenidas mediante cámaras. El objetivo del proyecto no es solo calcular el cambio de posición de una captura a otra, sino también estudiar los diferentes sistemas para relocalizar el coche en un mapa detectando detalles característicos en la imagen, permitiendo corregir la desviación acumulada. Entre las diferentes opciones para hacer esta relocalización, se desarrolla un sistema de segmentación y clasificación de carretera, ya que se considera que es la opción más interesante para el problema de corrección de la desviación.

LABURPENA

Dokumentu honetan kameren erabilerari esker lortutako irudiak erabilita ibilgailu autónomo baten posizioa lortzeko metodología ezberdinen azterketa burutzen da. Proiektuaren helburua irudi batetik hurrengorako posizio diferentzia kalkulatzea bakarrik ez da, baita mapa batean auto bat birlokaliztzeko sistema ezberdinen azterketa ere. Horretarako irudien xehetasun bereizgarriak detektatu egiten da, desbiderapen metatua zuzen daitzan. Birlokalizazioa lortzeko aukera ezberdinen artean, segmentazio sistema eta errepide sailkapen bat garatu da, desbiderapenaren zuzenketa lortzeko aukera hoberena baita.

eman ta zabal zazu

Universidad del País Vasco | Euskal Herriko Unibertsitatea

BILBOKO

INGENIARITZA ESKOLA

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

METHODOLOGY FOR CAR LOCATION AND MOTION DETERMINATION USING CAMERA SENSORS AND IMAGE SEGMENTATION AND CLASSIFICATION

MEMORY

Estudiante	*Ander Ramos Moreno*
Director	*Mikel Díez Sánchez*
Co-Director	*Asier Zubizarreta Picó*
Departamento	*Automática y control*
Curso académico	*2019-2020*

Bilbao, 12, Septiembre, 2020

INDEX

FIGURES INDEX

GRAPHS INDEX

TABLES INDEX

LIST OF ACRONIMS

- 2D: Two Dimensions
- 3D: Three Dimensions
- CNN: Convolutional Neural Network
- FAST: Features from Accelerated Segment Test
- GPS: Global Positioning System
- HLS: Hue-Light-Saturation
- IMU: Inertial Measurement Unit
- KNN: K-Nearest Neighbours
- LiDAR: Light Detection And Ranging
- RANSAC: Random Sample Consensus
- RGB: Red-Green-Blue
- RMS: Root Mean Square
- SURF: Speeded-Up Robust Features
- SVD: Singular Value Decomposition
- SVM: Support Vector Machine

1. MEMORY

1.1. INTRODUCTION

During the year, many people pass away on the road. The 90 % of the accidents that take place are due to human failure. Therefore, the automotive sector has been focused in the last decade on increasing the safety of the cars, not only by introducing different systems protecting the users from the impact, but also implementing systems such as the collision warning with auto brake as stated in [1], that help the driver avoid the danger and act if the driver does not react by stopping the car automatically.

The sector is clearly taking the direction of making high investments in developing an autonomous car that does not need human intervention to increase the safety on the road as much as possible. Of course, to achieve this objective, the car must incorporate many types of sensors, which substitute, in some way, the role played by humans when driving a car.

In this project, we will focus on analysing one of the most known and versatile sensors that are located in an autonomous car, the camera. With the appropriate use of it, the camera can detect not only the road, but also people or cars that are seen in an image, being faster and more reliable than a person. It can also relate different pictures and find out if we have passed through there before or how much we have moved from one picture to another.

Even if the camera can obtain many measurements, the key for effective autonomous driving is diversity of sensors and redundancy, overlapping sensors which can verify that what a car detects is accurate. These sensors include different sensors such as the LiDAR, IMU, radar or GPS. The fusion of all measurements leads to robust location and environment analysis of the car.

1.2. CONTEXT

In the Basque Country, the automotive sector represents over the 24 % of the PIB of the territory, being one of the most important ones. Many of the companies in the territory manufacture parts and develop projects for the different car brands that are in the market. Being part of the autonomous car development is key for keeping up with the requirements of the automotive world.

Knowing the real time position of the car is a fundamental factor for successful autonomous driving. When speaking about locating a car, our minds instantly thinks about the GPS (Global

Positioning System), however, the accuracy needed for this objective is much higher and the GPS is just a piece of a system composed by many other sensors.

Apart from global positioning sensors such as the GPS, local positioning sensors are needed. The sensor studied in this project, the camera, belongs to this last group. This project will be developed together with a doctoral thesis with the main objective of developing robust perception systems to deal with the uncertainties of the measurement system or failure of the sensors.

For making a perception system robust it is important to obtain the same measurement with different sensors, increasing the redundancy and in that way, being able to mitigate the effect of possible wrong measurements. An increase in the redundancy also has the objective of being able to stop the car safely when any of the sensors fails and stops working by using the measurements given by the others. Once the measurements are obtained, signals from different sensors are combined obtaining a robust car localization procedure.

To achieve this global objective, the first step is, based on the study of the most used implementation techniques, to develop perception algorithms that combine information of different sensors. This allows, by means of sensor fusion techniques, to define the road in a 3D environment and to position the vehicle in it. To carry out this localization, first, we need to understand which are the sensors used in an autonomous car, specifically, the GPS, the IMU, the radar, the LiDAR and the camera, in which is focused this project.

Probably the most known sensor for positioning is the GPS. The GPS is a radio-navigation system based on satellites that gives a global or local localization as long as there exists an unobstructed line of view between the device and at least four GPS satellites.

The LiDAR perceives the environment by means of millions of light pulses emitted per second according to a well stablished pattern. Those pulses are reflected by the objects surrounding it in a period of time called Time of Flight, based on how far they are from the sensor. In that way, it is possible to represent a dynamic environment in 3D according to the point cloud that is obtained thanks to the rotative shaft of the LiDAR.

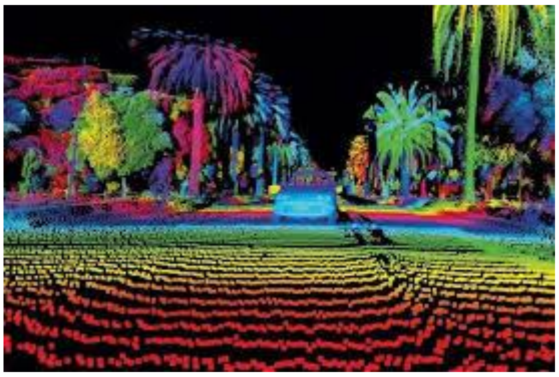


Figure 1: LiDAR detection

The radar works in a similar way as the LiDAR but using electromagnetic waves instead of light pulses. In general, the electromagnetic waves are less accurate than light pulses and can not detect small objects clearly, having also reflection problems with some objects. However, the radar is much cheaper and, normally, it can work better at night or in bad weather conditions. In addition, the range of the radar is in general higher than the one of the LiDAR, being also faster in terms of information transmission, which can be extremely important at high velocities. However, for the car localization problem, these factors are not so important as accuracy, making the LiDAR the best sensor for it.

The IMU is composed of a combination of accelerometers, gyroscopes and magnetometers that allow it to measure the velocities, angular accelerations and magnetic fields of the bodies where it is placed. It is commonly used together with the GPS for determining the position when connection of the GPS is lost due to tunnels or electronic interference. However, one of its main disadvantages is that the error accumulates over time, thus having to correct the estimated position from time to time.

The camera uses a lens that takes all the light rays bouncing around and uses glass to redirect them to a single point. When all of those light rays meet back together on a digital camera sensor, they create a sharp image. The project will be focused on obtaining measurements with the camera, although other sensors might be used to add information or for validation.

Figure 2: Camera for autonomous car

Our search for redundancy in the measurements, implies that we have to obtain the same measurement with as many sensors as possible. In the next step of the doctoral thesis that is being developed in parallel, the measurements of the previously described sensors will be fused to obtain a robust position estimation. In this project, we will focus on obtaining measurements only from the camera and use the measurements of the other sensors to compare the results.

To understand the objective of the project, it is important to be familiar with the reference system of a car and the reference system of a camera. In a car, the x axis points forward, the y axis points to the left of the car and finally, the z axis necessarily points up. The rotation along the x axis is called roll, and along the y axis is called pitch. These magnitudes represent how much the car tilts when cornering, braking and speeding up, being a general expression of cars stability. The rotation along the z axis is called yaw, and represents the turn of the car. In this project, although we will manage to calculate the three rotations appearing in the car, we will be interested just in the last one, being the one giving information about the car orientation parallel to the road surface.

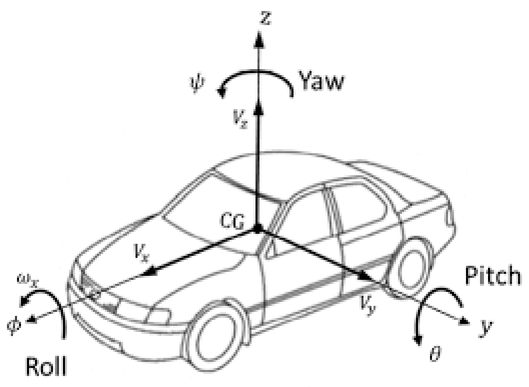


Figure 3: Car reference system

The camera will have a different reference system with the z axis pointing outwards of the objective, the y axis pointing downwards and the x axis pointing to the right. As the cameras are placed at the top of the car pointing forward, when obtaining the corresponding measurements, it will be needed to perform a change of the reference system to adapt the results to the car.

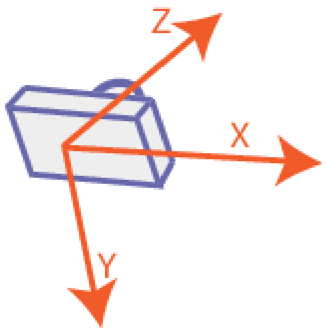


Figure 4: Camera reference system

Regarding the images obtained from a camera, there are some notions that are important to know about images to understand what a computer vision software does with it.

An image is composed by pixels, which are small coloured squares joined one to each other. The more pixels an image has, the better the resolution is. When a software with a computer vision module opens an image, each pixel of the image is defined by its location (according to the height and wide of the image) and by its color.

Depending on the type of image we are working with, the color will be defined in a different way. The most typical type of color image is the RGB, that stands for Red, Green and Blue. As its name says, each pixel is defined by a mixture of these three colors, which intensity is determined by a rank that goes from 0 to 255. Apart from the RGB, there are other types of color image types such as the HSL, that stands for Hue, Saturation and Lightness. The simplest type of image is the grayscale one, where the color is just defined by a single number that ranges from 0 to 1000, where 0 is black and 1000 is white. The grayscale image is the most used type of image for implementing computer vision algorithms.

1.3. OBJECTIVES AND SCOPE OF THE WORK

The main objective of the project is to obtain the position, understanding position as the location (position) and orientation (yaw), of the car as accurate as possible by means of cameras. This will allow the sensor fusion algorithms to better estimate the actual position in the presence of noise or other sensors malfunction.

To carry out this objective, the first step is gathering all the developed methods for calculating the motion of a camera based in the images taken by it. Among all the information, the appropriate selection of the methods to be developed must be done, ensuring that they can obtain the appropriate results.

Then, a program must be developed with the capacity of obtaining as much information as possible of the car position from as less images as it can. The camera position will consist of the position of the camera in the cartesian coordinates (x,y,z) but also of the rotation in the z axis, which corresponds to the rotation of the car (yaw).

A secondary objective of the project is finding a method for obtaining useful information for the localization of the car, such as, the geometry of the road the car is at or traffic signals that indicate an event that can be seen in a map. The methods will be used to correct the error accumulated over time by the previous localization process when a place or characteristic object is recognized.

To fulfil this task, first, a study of the different detection and recognition technologies in images will be done. After studying the different methods, the one that could obtain more information will be developed and tested, leaving the development of the other studied methods for a later project.

1.4. BENEFITS IT BRINGS TO WORK

Being able to obtain the yaw variation and displacement of the car with the camera will add more information to the sensor fusion for estimating the position, increasing the redundancy and reducing the error. This brings more robustness to the system in case of failure of other sensor.

It also will contribute to the community in the development of the autonomous car, being this one of the main goals of engineering in the following decade.

1.5. REQUIREMENT DESCRIPTION AND ANALYSIS OF THE STATE-OF-THE-ART

1.5.1. DATABASE SEARCH

As we do not have the possibility of creating our own database of images by means of a car with coupled sensors, the first step is to look for the available databases for developing the project.

In the Astyx HiRes2019 Dataset, as stated in [3], we found a dataset containing 500 frames obtained by means of a color camera of 30 Hz with a resolution of 2048 by 618 pixels, together with a 13 Hz radar and a 10 Hz LiDAR. Despite of the difference between the frequencies of the sensors, the 500 frames are synchronized in advance and have over 3000 3D object annotations. The work proposed in the paper of the dataset, is focused on obtaining high resolution radar data, however, cameras are also used to estimate the relative position of the car, based on least squares approximation and applying the RANSAC algorithm for outlier rejection. It also performs detection of different objects on the road (mainly cars) by means of deep learning algorithms using not only the radar, but also the camera and the LiDAR.

In the Berkeley DeepDrive dataset, as stated in [4], we found a dataset containing 100,000 video sequences involving different times of the day, weather conditions or driving scenarios. The images contain lane annotations together with bounding boxes annotations for different categories and some pixel-level annotations. The dataset is meant to be used for fulfilling seven tasks that are needed for successful autonomous driving, which are: image tagging, object detection, lane marking, drivable area detection, semantic segmentation, multiple object tracking and imitation learning.

As stated in [5], the Level 5 dataset owned by Lyft is public and contains over 100,000 hours of video sequences with the measurements of three 10 Hz LiDAR devices, one on the top of the car and the other two placed near each of the front wheels. It also contains the measurements of six 360 degrees cameras that are synchronized with the LiDAR devices. The objective of the dataset is to join forces to accelerate the development of autonomous driving by sharing the data obtained by their car fleet. They also share their knowledge in form of online webinars and competitions.

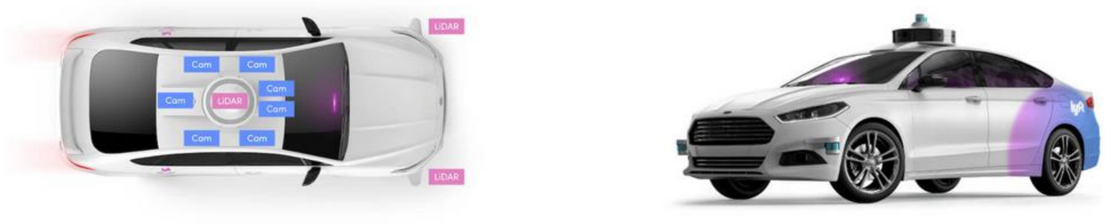


Figure 5: Level 5 car for data collection

The nuScenes dataset, as stated in [6], contains the measurements of one LiDAR, 5 radars, 6 color cameras, an IMU and a GPS for a total of 1,400,000 images. The cameras are of 12 Hz and have a resolution of 1600x900 pixels. The LiDAR has a frequency of 20 Hz while the radars have a frequency of 13 Hz. The IMU and GPS have a frequency of 1000 Hz. It also has manual annotations for 23 different object categories. The dataset differentiates from other existing datasets due to the 360 degrees coverage of its sensors and for having data obtained at night time and rainy conditions. Car localization is performed by creating a map from the obtained points from the LiDAR. Then, MonteCarlo Localization scheme is applied by using the LiDAR and camera measurements.

The Oxford Radar RobotCar dataset, as stated in [7], is focused on validating the measurements obtained with the Frequency-Modulated Continuous-Wave class of radar, allowing robust detection in adverse environmental conditions. Apart from the radar, the dataset contains the measurements of two 3D LiDAR devices and another two 2D LiDAR devices, a stereo camera, three monocular cameras and an IMU/GPS system. There is a total amount of 280 kilometres of urban driving in different weather conditions.

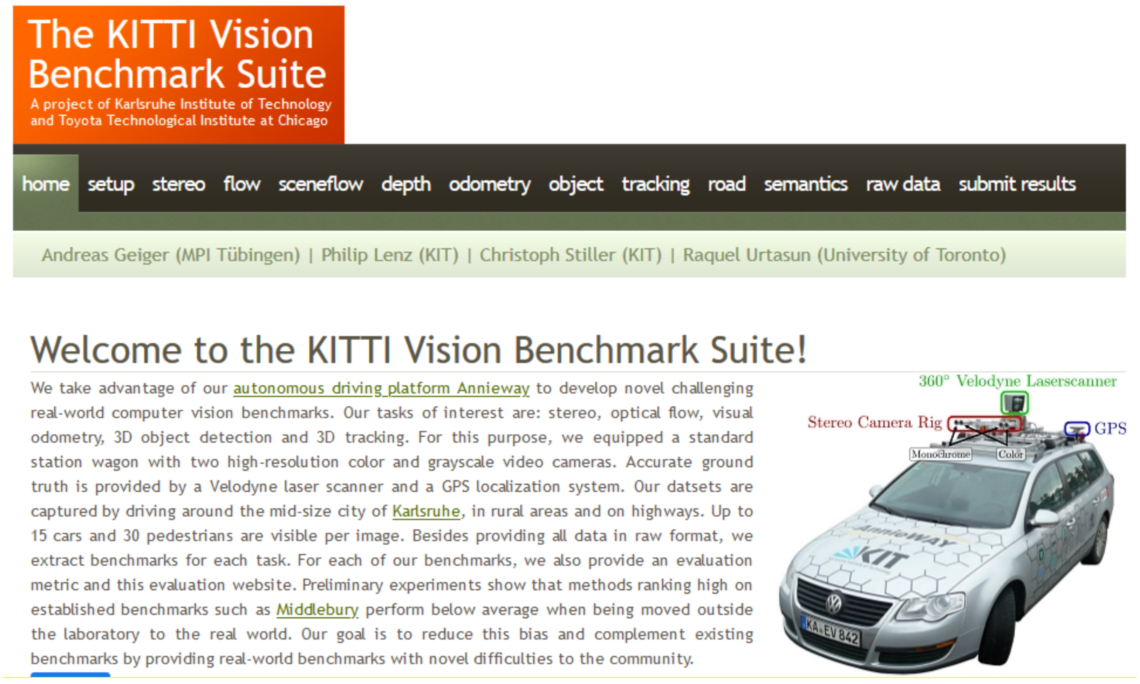
The PandaSet, as stated in [8], is made available for both academic and commercial use. It contains, apart from the GPS and IMU measures, 48,000 camera images, 16,000 LiDAR sweeps, obtained from 6 cameras and 2 LiDAR devices. It also makes available the labels for 37 categories applying semantic segmentation and 28 categories applying object detection. The objective of the PandaSet is to promote and advance in the research of autonomous driving and machine learning by making their information public.

Figure 6: PandaSet car for data collection

In the KITTI Vision Benchmark Suite, as stated in [2], it can be found a project by the Karlsruhe Institute of Technology and the Toyota Technological Institute at Chicago. This benchmark provides a lot of different datasets of raw sensor measurements, among which it can be found the images obtained by four cameras, of which, two are grayscale cameras and the other two are color cameras. The image sets consist of different scenarios that include city, residential and road environments, including the corresponding calibration parameters for each dataset. The images have a height of 375 pixels and a wide of 1242 pixels.

Apart from the images obtained by the four cameras, the measurements corresponding to the LiDAR (Light Detection and Ranging), the IMU (Inertial Measurement Unit) and the GPS. The LiDAR and the cameras work at 10 Hz while the rest of the sensors work at 100 Hz.

The set contains a total amount of six hours of traffic scenarios, which suppose a total number of 216,000 images, with the objective of providing adequate data to develop multiple tasks necessary for successful autonomous driving. Among these tasks it can be found stereo vision, object detection or optical flow with the appropriate labels for developing the corresponding Convolutional Neural Network if needed.



The KITTI Vision Benchmark Suite
A project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago

home setup stereo flow sceneflow depth odometry object tracking road semantics raw data submit results

Andreas Geiger (MPI Tübingen) | Philip Lenz (KIT) | Christoph Stiller (KIT) | Raquel Urtasun (University of Toronto)

Welcome to the KITTI Vision Benchmark Suite!

We take advantage of our autonomous driving platform Annieway to develop novel challenging real-world computer vision benchmarks. Our tasks of interest are: stereo, optical flow, visual odometry, 3D object detection and 3D tracking. For this purpose, we equipped a standard station wagon with two high-resolution color and grayscale video cameras. Accurate ground truth is provided by a Velodyne laser scanner and a GPS localization system. Our datsets are captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image. Besides providing all data in raw format, we extract benchmarks for each task. For each of our benchmarks, we also provide an evaluation metric and this evaluation website. Preliminary experiments show that methods ranking high on established benchmarks such as Middlebury perform below average when being moved outside the laboratory to the real world. Our goal is to reduce this bias and complement existing benchmarks by providing real-world benchmarks with novel difficulties to the community.

Figure 7: KITTI benchmark

1.5.2. Position estimation algorithms

Once the dataset study is done, a search in the literature for position estimation methods must be done.

Most developed algorithms regarding image processing for camera position estimation are described on the book called “Multiple-view geometry in computer vision”, as stated in [9]. In it, we can find theoretical explanations of how to analyse scenes having different number of images of them from different points of view. In this project we will focus mainly in the analysis of scenes from two points of view, also called stereo vision.

In the chapter describing the stereo vision concept and among utilities of relating two images and what we can get with them, appears the position estimation by feature matching applying the RANSAC algorithm for outlier rejection. There are some subchapters showing step by step the development of the full method and the options it leads to in the position estimation subject.

However, it is not the only method to obtain the car position. In [14] a simple triangulation method is described to find the relative position from the location of a pair of cameras to the objective car in the image. This method, even if it is far from our objective, can be modified to instead of calculating the distance to a moving object (in this case, a car), the position to any fixed object in the image. If the position is again calculated to that same object in the following frame, the relative displacement of the car is obtained.

In [15], it is described a system based on a monocular camera and an IMU that can make a 6 degrees of freedom position estimation. The system is used to recognize an image that has previously been seen and relocating the car based on the location of that image. In addition, it also uses the method to calculate the relative pose between one frame to another, which is the objective of our project. This is done by calculating the fundamental matrix that relates each image pair and applying the RANSAC algorithm to reject the possible outliers. This method for position estimation was also proposed in [3] and is explained in [9]. This system can be found also developed in other papers such as [16], where the camera and the IMU are fused once again to estimate the car location.

1.5.3. Study of the methods for relocalization

The second objective of the project is the study of the different methods to extract valuable information from the image such as traffic signals, road shape or characteristic features or objects. First of all, the methods for detecting objects or parts of the road must be found so we can select the appropriate one.

The most used applications for this purpose are based on deep learning, which are algorithms of automatic learning. Among the different types of architectures for deep learning, the CNN (Convolutional Neural Network) is the most appropriate one for image processing. To obtain the desired CNN for our application a training process must be carried out by using several labelled images. If the training is successful the network is able to replicate the process of detection by itself in images that are different from the ones of the training set.

The most direct way of using deep learning for our objective is using it for object detection, which allows to detect and mark objects in an image, introducing them in a bounding box. This technology can be very useful, as stated in [10], not only for detecting objects such as cars, people or signals, but also for finding its position and dimensions using the LiDAR and camera measurements. In the paper it is proposed a multi-view 3D object detection network that predicts the 3D bounding boxes using a LiDAR front view, a LiDAR birds-eye and an RGB image as inputs.

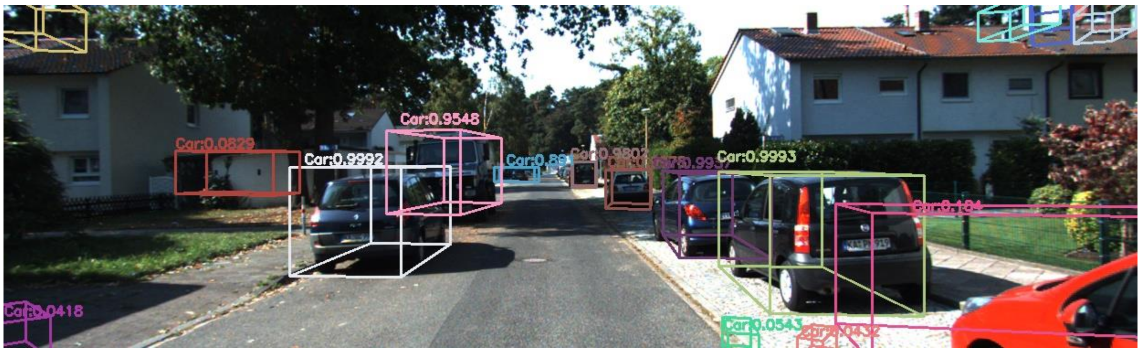


Figure 8: 3D object detection

If the extension of what we want to detect is larger or needs to be detected in a more precise way than introducing it in a bounding box, a semantic segmentation network could be trained, which allows to make a pixel-level classification of the image. The possibilities of this method are bigger as enables the detection of bigger instances such as buildings or the road and, based on its shape, instances with complex geometry such as people or cyclists. In [11] it can be found an experimental study of the application of different segmentation networks in a driving scenario. Both accuracy and training time are extracted as the most important results and parameters to consider when choosing a CNN.

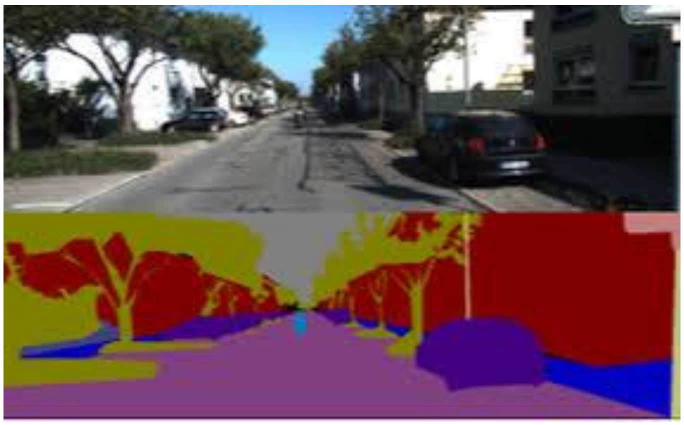


Figure 9: Semantic segmentation

Apart from the applications of deep learning algorithms described before, there are many others such as simple classification or using it to perform regression. This last one is also a very useful tool for autonomous driving applications, as stated in [12], where a regression CNN is used for detecting the lanes in the road. In this post, the concept of transfer learning is introduced, being a very interesting method to save work and time. Transfer learning consists of retraining an already trained network with many images with our training set, which leads to less computation and work as we do not have to build a customized network for our application.

However, the detection of features in an image is not only possible by applying deep learning algorithms, there are other methods, that have the advantage of avoiding the training step. As stated in [13], an RGB image can be converted to HLS (Hue-Light-Saturation), which allows us to isolate the pixels in an image with a determined color despite of the lightness. By applying it we can isolate the characteristic color of the lines in the road and then identify the shape of the lines by using the Hough transform. This method could also be applied for the recognition of objects of known color and shape such as traffic signs.

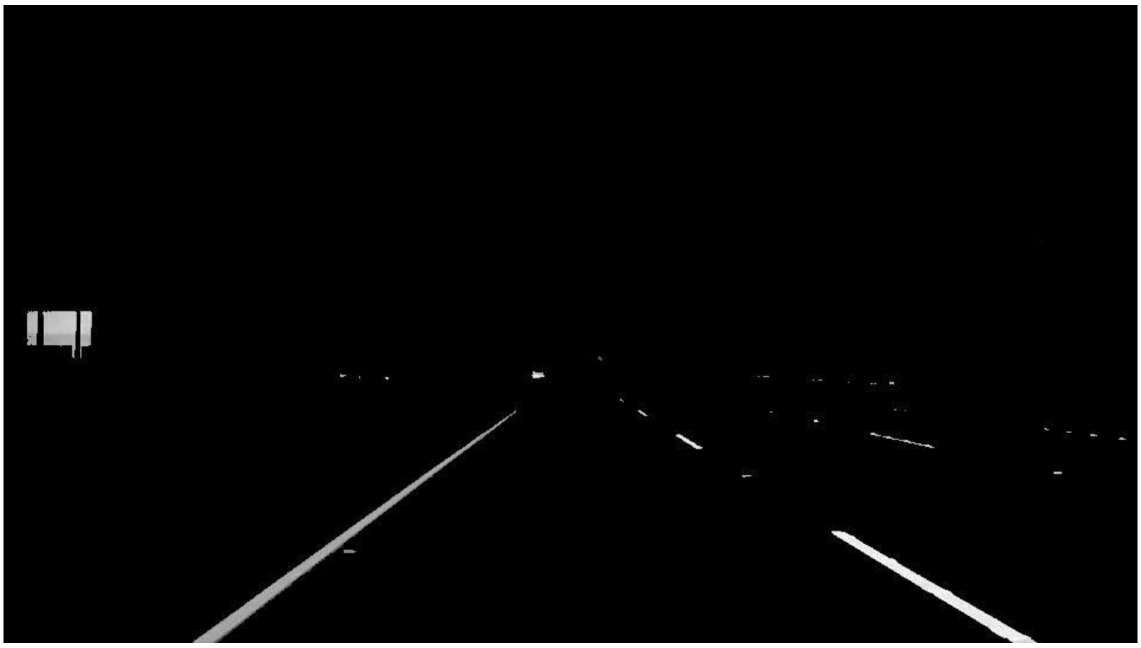


Figure 10: Image of road after isolation of white color

Once the research of the existing detection methods has concluded, an investigation of the recognition or classification methods must be done. As mentioned before, both object detection and segmentation networks have the ability of classifying the instances they detect, however, there are other classification methods that might be better and, therefore, have to be considered.

Support Vector Machines (SVM) can be used as classifiers, but also for regression or other tasks such as outlier rejection. They are supervised learning models that, given a set of training examples, each marked as belonging to one category, a training algorithm builds a model that assigns new examples to one category or the other. As it can be seen, the steps to generate the classification model is similar to the one required to train a CNN, however, the computation time required to train the SVM is much lower.

The data used for training the model is used to define a hyperplane that determines in which category falls a new example. In figure 11 it can be seen how the hyperplane is defined according to the best margin from the training data. In the example, a linear hyperplane (linear SVM) can be seen, but the order of the hyperplane can be increased defining, for example, a cubic SVM.

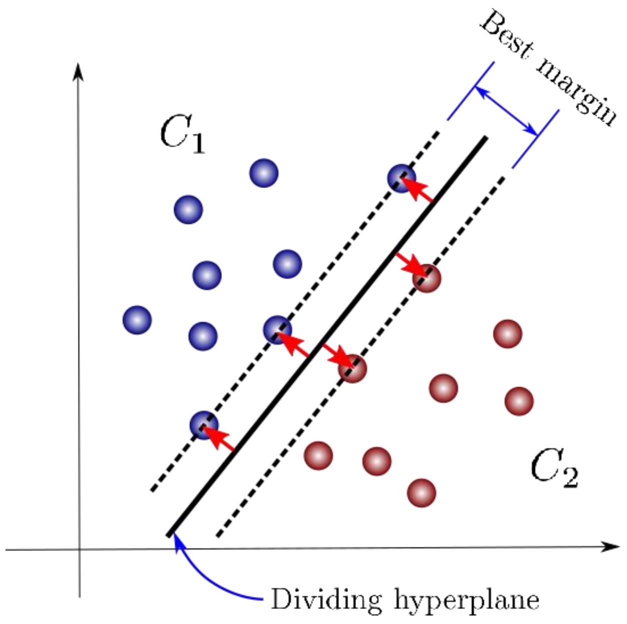


Figure 11: SVM for classification

The K-Nearest Neighbours (KNN) algorithm can be used for classification or regression, being a very simple method in terms of how it works. When a new example is introduced in the model, the K closest training examples are analysed and seen which category they belong to, assigning the most appearing category to the new example. The value of K, representing the neighbours that are considered, is determined according to the type of KNN algorithm, for example, a Fine KNN only considers the closest neighbour for determining the category. Like the SVM, the time needed for training a KNN model is much lower than the one needed for a CNN.

1.6. Description of the proposed solution. Design

In this section, we will analyse the different alternatives discussed in the state-of-the-art section and choose the ones that best fit our objectives. First, among the proposed datasets the most appropriate one for the project will be chosen. Then, a method for obtaining the car position by means of the camera will be defined and finally, the development of a model with the ability of extracting information from a driving environment will be defined.

1.6.1. DATASET SELECTION

The first step is the selection of the database to use. It is considered that all the studied datasets have enough data for developing the required tasks. However, the Astyx HiRes2019 Dataset [3], contains just 500 frames, which, even if seem to be enough for the development, might not be enough for the validation.

Among the options, the nuScenes dataset [6] and the Oxford Radar RobotCar dataset [7] are focused on validating the radar performance in bad visibility conditions, which is far from our scope of the project. The KITTI Vision Benchmark Suite [2], is the selected one due to several reasons that make it stand over the rest of the datasets. The main factor for its selection is its use in the parallel tasks that are carried out in the doctoral thesis project. The use of a different datasets could lead to problems in the compatibility when fusing the measurements of the sensors or the necessity of modifications of the algorithms. Apart from that, the KITTI dataset contains measurements obtained with both color and grey cameras, being the color cameras necessary for the second objective of the project but not as good as the grey ones for the first part due to the less contrast of their pictures. It also has similar resources to the other datasets in terms of material for the development of different autonomous driving tasks such as object detection or semantic segmentation.

1.6.2. SELECTION OF LOCALIZATION METHOD

Next, the selection of the method to localize the car is done. The matching method is selected with outlier rejection by using the RANSAC algorithm used in [15], due to the detailed description available in [9] and the good results obtained by it in some of the previously mentioned papers. The disadvantage of the method is that for obtaining the displacement of the car, it is necessary to use the measurements obtained by the IMU, which reduces, in part, the redundancy of the system, being this one of the requirements of the project.

The discarded options include the triangulation algorithm used in [14] because even if it is easy to find the position difference of the car, it is necessary to develop previously a detection algorithm, being this the hardest part of the process. In our case, to use this method, would imply the development of a CNN for detecting a fixed point and the development of some method to make sure that the point we are detecting is the same in all images. In addition, the depth estimation is one of the highest limitations of a camera, obtaining, in general, inaccurate results.

Also, the Monte Carlo localization algorithm described in [6] is an interesting option due to the accuracy it has (10 cm error), however, the inputs of the algorithm are composed, apart from the images of the camera, of a point cloud obtained by the LiDAR and a map. The necessity of using the LiDAR for position estimation would reduce the redundancy of the system, reducing the data for the sensor fusion. Anyway, in this project we do not have any map available, so the development is not possible.

1.6.3. Selection of a method for relocalization

After studying the existing methods for detecting instances for relocalization on the road, it has been decided to develop a system able to detect the road and classify it according to the instance the car is at (straight, crossroad, left curve, right curve). One of the main reasons for selecting this option is the data provided by the KITTI dataset for building a CNN to segment the road. As explained before, road detection can only be achieved by using this type of CNN due to the size occupied by the road in the images. The KITTI dataset provides a set of labelled images for training a CNN to segment both the road and the lane the car is at.

It is also possible to obtain a classification CNN that analyses the raw image and obtains directly the category of the image. The problem is that we would have to label all the training set and gather a big number of images. This is due to the fact that most of the instances on the road correspond to straights, being necessary for a good training to have a balanced number of training information per category.

It is decided to train the road segmentation network with the KITTI data and afterwards, postprocess the obtained information and feed it to a classifier. By doing this, we have the possibility of removing the noise that might appear in the output of the segmentation network. As the training data for the classifier is very little to train a classification CNN, it is decided to train a simpler classifier, such as an SVM or a KNN, which do not need a big amount of training information to perform well and require a very reduced time to be trained.

Methods like the one described in [13], are not chosen due to the lack of robustness compared to a segmentation CNN and also because if the training set is given, they require, the same, or even more work than the selected option for its development.

Finally, a software that incorporates all the previously mentioned algorithms must be selected. The available programs for doing it were MatLab and Python. MatLab was chosen due to the higher knowledge of the user of the program and due to the big amount of applications incorporated in the program for visual analysis and autonomous driving tasks. Additionally, it is chosen for compatibility with the doctoral thesis developed in parallel which uses MatLab.

2. METODOLOGY CARRIED OUT IN THE WORK DEVELOPMENT

2.1. TASKS, PHASE, TEAMS AND PROCEDURE DESCRIPTION

- Task 1: State-of-the-art study.
  - ✓ The different methodology carried out in the past for processing images will be studied together with the available types of camera that can be used for effective autonomous driving. The available image databases will be searched and analysed. The methodology for locating a car given a set of images will be studied in detail.
  - ✓ Work load: 75 hours.
  - ✓ Duration: 3 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer.

- Task 2: Method choice.
  - ✓ Among the different options, a determined set of cameras together with the proper database are selected. The methodology for processing the images is defined and the algorithms to be use in the car localization are chosen and adapted for the project.
  - ✓ Work load: 25 hours.
  - ✓ Duration: 1 week.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer.

- Task 3: Position algorithm development.
  - ✓ An algorithm for the position estimation is developed with MatLab taking into account the selected methodology and by using the proper resources found in the previous tasks.
  - ✓ Work load: 50 hours.
  - ✓ Duration: 2 weeks.
  - ✓ Human resources: Control engineer.

- ✓ Technical resources: Computer, MatLab license, computer vision toolbox license, image processing toolbox license.

- Task 4: Testing the algorithm and obtaining results.
  - ✓ Once the algorithm is built it is tested and in case of errors, fixed. When a proper set of results according to a database are obtained, they are validated by visual analysis and by comparison with the measurements of different sensors.
  - ✓ Work load: 50 hours.
  - ✓ Duration: 2 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer, MatLab license, computer vision toolbox license, image processing toolbox license.

- Task 5: Development of the segmentation CNN
  - ✓ From the selected database an adequate number and types of images are selected for the training of the network. The most adequate type of pretrained network is chosen and code for the corresponding training is written.
  - ✓ Work load: 50 hours.
  - ✓ Duration: 2 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer, MatLab license, computer vision toolbox license, image processing toolbox license, deep learning toolbox license.

- Task 6: Postprocessor development.
  - ✓ The results given by the segmentation CNN are processed to remove the possible noise in them. Then, data is extracted from the segmented images to obtain a set of parameters able to describe the images.
  - ✓ Work load: 50 hours.
  - ✓ Duration: 2 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer, MatLab license, computer vision toolbox license, image processing toolbox license, deep learning toolbox license.

- Task 7: Classifier development.
  - ✓ A set of adequate images is selected for training a proper classifier. The parameters from the training set are extracted from the training images as defined in the previous task and used for obtaining the classifier.
  - ✓ Work load: 25 hours.
  - ✓ Duration: 1 week.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer, MatLab license, deep learning toolbox license.

- Task 8: Testing the classifier and obtaining results.
  - ✓ Multiple datasets are chosen for validation, extracting the data from the corresponding images and introducing it in the potential classifiers for determining which ones are successful and validating the results.
  - ✓ Work load: 50 hours.
  - ✓ Duration: 3 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer, MatLab license, computer vision toolbox license, image processing toolbox license, deep learning toolbox license.

- Task 9: Writing the documentation.
  - ✓ Once the calculations of the project are done and the results are validated, being sure that everything works as expected, the documentation of the project is written showing the complete methodology used and the results obtained with it.
  - ✓ Work load: 75 hours.
  - ✓ Duration: 4 weeks.
  - ✓ Human resources: Control engineer.
  - ✓ Technical resources: Computer.

2.2. GANTT DIAGRAM

For completing the project successfully, a planification of the tasks has been done taking into account the available time. In this section, the previously presented tasks will be given dates of start and end, which will allow to build the Gantt diagram of the project, as it can be seen in table 1.

		April				May					June				July				August		
Task	**Duration**	3-10	10-17	17-24	24-01	01-08	08-15	15-22	22-29	29-05	05-12	12-19	19-26	26-3	3-10	10-17	17-24	24-31	31-7	7-14	14-21
State-of-the-art study	3 weeks	🟦	🟦	🟦																	
Method choice	1 week				🟦																
Position algorithm development	2 weeks					🟦	🟦														
Testing the algorithm and obtaining results	2 weeks							🟦	🟦												
Development of the segmentation CNN	2 weeks									🟦	🟦										
Postprocess development	2 weeks											🟦	🟦								
Classifier development	1 week													🟦							
Testing the classifier and obtaining results	3 weeks														🟦	🟦	🟦				
Writing the documentation	4 weeks																	🟦	🟦	🟦	🟦

Table 1: Gantt diagram

As it can be seen the project begins the 3rd of April and finishes the 21st of August, as it is planned to be done during the second semester of the course. The tasks that are done during the months of July and August (the last two tasks) have an additional duration, in spite of the work load they require. This is due to the fact that they are done during the summer, when there are several periods of holidays.

The remaining time until the 10th of September, when the project is presented, is left to face the possible unforeseen that might appear during the project.

2.3. CALCULATIONS AND ALGORITHMS

In this chapter, it will be explained step by step how the design process has been performed. There are two clearly differentiated parts of the project: the position estimation algorithm and the design of the classifier.

2.3.1. POSITION ESTIMATOR

Our objective in this part of the project is to find a way of obtaining the camera movement (which we assume equal to the car movement) from one frame to its consecutive one. There are several studies explaining how two 2D views where the same point appear can be related, this is called epipolar geometry, and depends only in the characteristics of the cameras and its relative position.

As explained in the context, we are working with two grayscale cameras and another two color cameras. As our study is based on geometry, color does not matter and for this calculation the color images will be converted to grey. By doing this, we are able to use four independent cameras to obtain the same measurements, adding redundancy to the system and making it more robust in the sensor fusion phase.

2.3.1.1. The definition of the problem

The problem could be described as a three variable problem, where the variables are the camera parameters, the relative position of the cameras and the scene. By knowing two of these variables it is possible to obtain the third one.

If we had the scene and the relative position and wanted to obtain the camera parameters, we would be facing a calibration methodology, where a completely known chess board is used as the scene and we set the cameras at known positions.

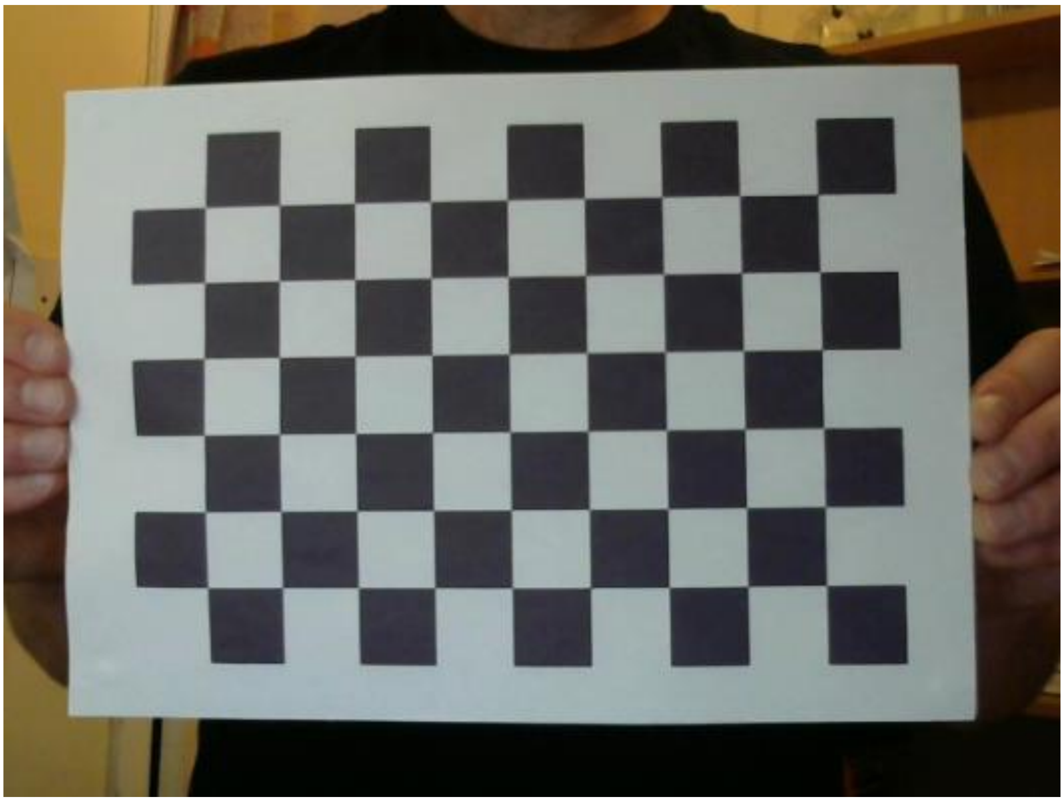


Figure 12: Camera calibration chessboard

In our case, all calibration parameters are given by the KITTI database and we do not have to go through this step.

If the camera parameters and the relative position of the cameras were known we would be facing a scene reconstruction problem. In fact, this approach can be done with our system and is called stereo view, or what is the same, 3D view. Exactly as it happens to us, humans, we need a pair of eyes to perceive the world in three dimensions, therefore, to reconstruct a scene we also need two 2D views. As we have a total of four cameras available, this reconstruction would be possible, however, it is a complex process and it is not the objective we are chasing.

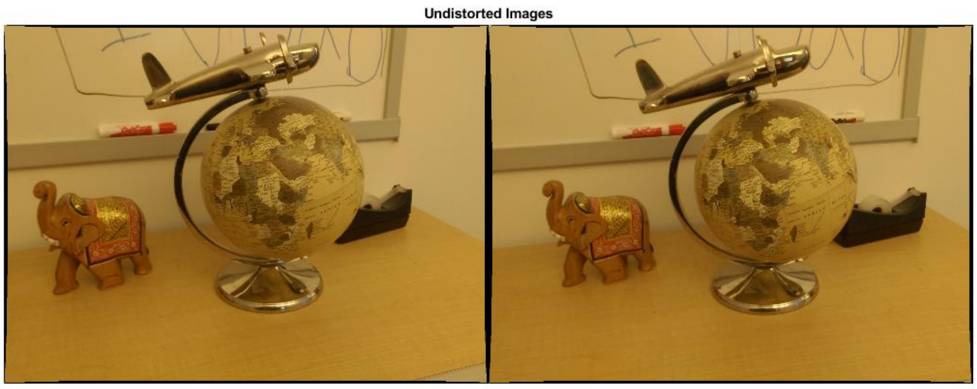


Figure 13: 2D image pair

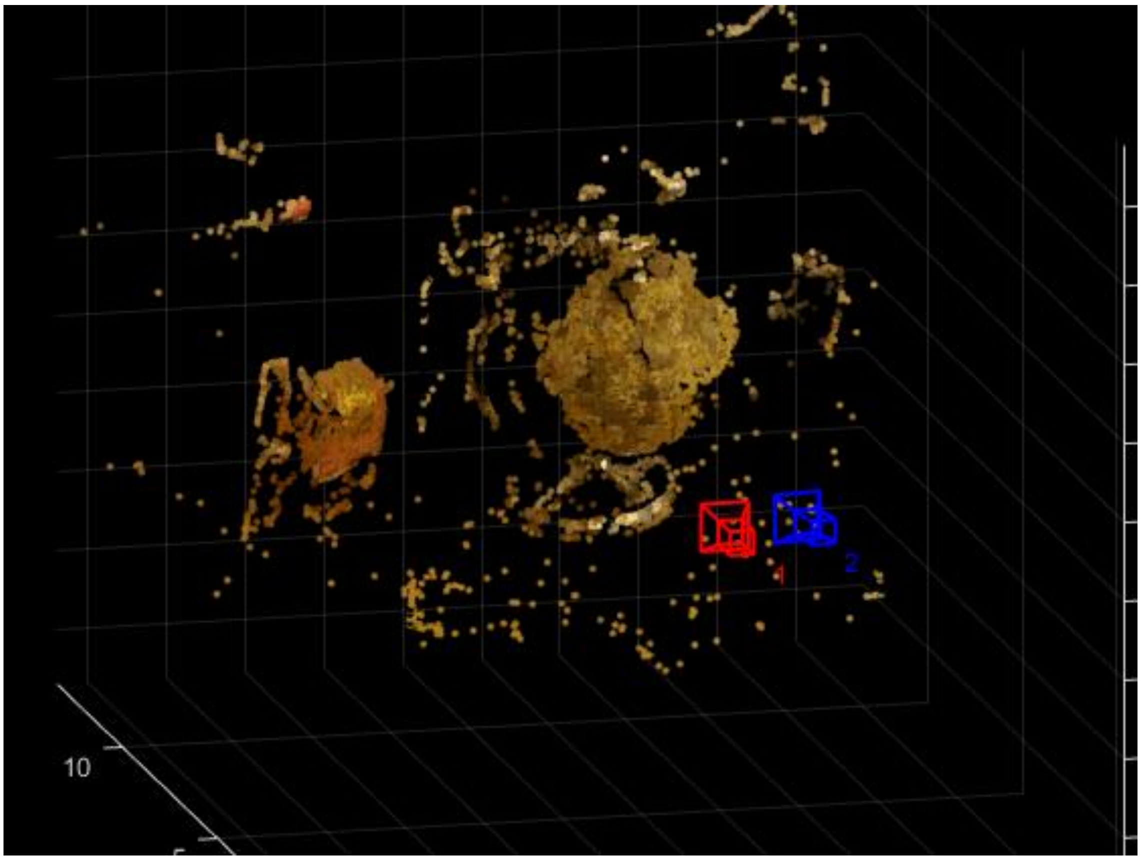


Figure 14: Scene reconstruction

The third possibility is to calculate the relative position of the cameras by knowing the camera parameters and the scene, which, using a single camera in motion at two different time frames, is the same as calculating the camera position change, which is our goal.

Of course, we do not need to know every single point of the scene and its location in the real world, which will make the process too complex. It is enough to recognize some "pairs" of points in the images, this is, the projection in the two 2D images of a real world point. This method is called matching, and it will provide empirical information to find a relation between the two images in the form of a matrix called the fundamental matrix, as it will be explained later.

2.3.1.2. Feature detection and matching

For detecting a pair of points, as the scene is not known dimensionally speaking, we would need to match these points by performing a visual approach, which is the same as recognizing the same point in both images. First, we need to detect corners or features in the image. We define corners as points whose neighbour pixels stand in two dominant and different edge directions, a better understanding can be reached by looking at figure 15.

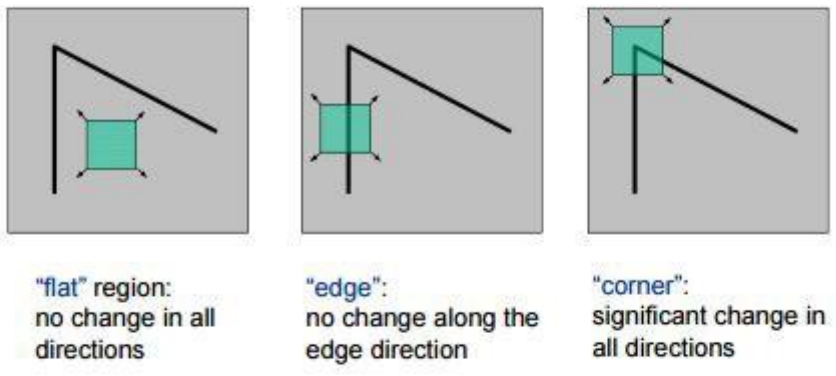

"flat" region:
no change in all directions
"edge":
no change along the edge direction
"corner":
significant change in all directions

Figure 15: Corner descriptor

Features refer to boundaries between groups of pixels such as sudden changes in color or geometry of objects and, in general, corners are a part of them. MatLab provides different algorithms to detect the features in an image. Each one has a different way of measuring the contrast of a determined pixel and its neighbours. It is chosen the SURF (Speeded-Up Robust Features) detection algorithm due to the better results obtained experimentally against other feature detectors such as FAST (Features from Accelerated Segment Test) or the minimum eigenvalue feature detection algorithm. Once the features of the pair of images is detected, the matching process is carried out, joining the features of each of the images together.

However, among these matchings, there will be some wrong ones, which implies that points that are matched together might not represent the same instance in both images. These wrong matchings are called outliers, while the right ones are called inliers. In figure 16 two consecutive images taken from a car are shown overlaid with the features corresponding to each image (red circles and green crosses) detected and matched with each other. Two outliers can clearly be seen marked in red as they are matching points that are far away from each other in the images.

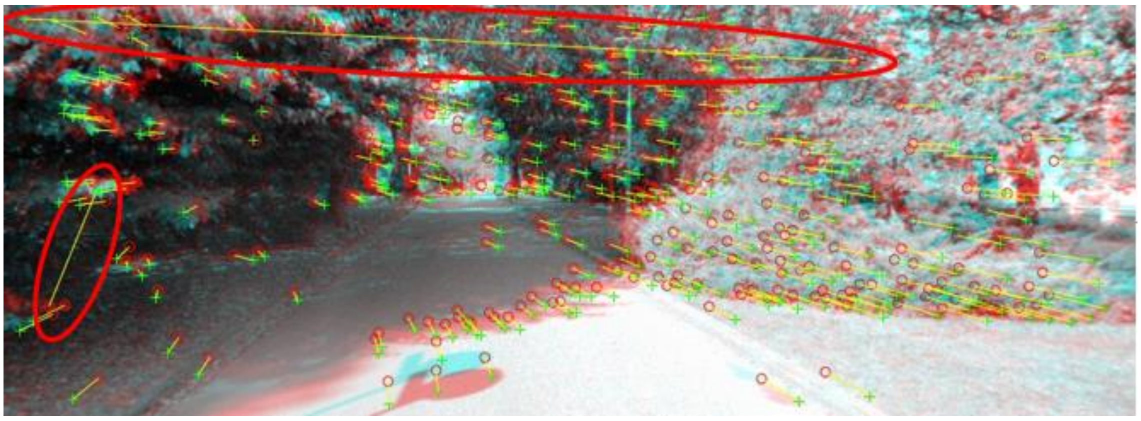


Figure 16: Matching of two consecutive frames

2.3.1.3. RANSAC algorithm for outlier rejection

The appearance of outliers is a normal thing in the matching process, as there are many different points in an image that might have similar characteristics regarding the contrast properties of their neighbour pixels. The rejection of outliers is a process with high importance as the use of an outlier in the calculation will derive in a wrong solution of the problem.

The main process for rejecting outliers is the use of the RANSAC algorithm. For a better understanding of how the algorithm works it will be explained for its use in the simple case of approximating a straight line with a set of points.

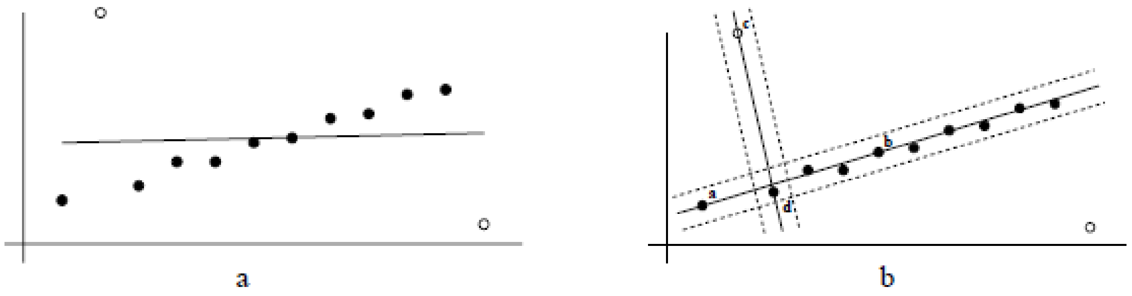


Figure 17: RANSAC algorithm applied to straight line approximation

In figure 17, the solid points would be the aforementioned inliers, while the open ones would be the outliers. In graph a it can be seen that if the outliers are not rejected, high deviation in the final result appears. The RANSAC algorithm randomly selects pairs of points and joins them with a line, defining afterwards a threshold distance from it. If the number of points within the defined threshold is low (2 for case c-d) it means that at least one of the points is an outlier, in this case, c. On the other hand, if the number of points within the threshold is high (10 for case a-b), it means that the points are inliers.

A curious characteristic of the RANSAC algorithm is that, unlike other approximation algorithms that try to use as much information as possible to obtain the solution and afterwards try to eliminate the invalid data, this method does the opposite. It estimates a solution with as small data as it can for the solution to be feasible and then enlarges the set with consistent data if possible.

2.3.1.4. Obtaining the fundamental matrix

Once we have done the outlier rejection, we need to translate the matched pair of points into an algebraic expression. This algebraic expression is what we call the fundamental matrix, a 3x3 matrix that relates the points of an image with its matches. The fundamental matrix satisfies the following expression:

$$\boldsymbol{x}'^{T} \cdot \boldsymbol{F} \cdot \boldsymbol{x} = 0 \tag{1}$$

Where F is the fundamental matrix, x is a point in one of the images and x' is the matched point in the other image. By using this expression we will be able to estimate the fundamental matrix. It can be clearly seen the importance of the outlier rejection step, as the use of an outlier for estimation would mean introducing a “false” pair of points in the equation and therefore the obtention of a wrong result.

As x and $x'$are points of the form (x,y,1), each point match arises one linear equation including the unknown entries of F. If we develop the previous equation we obtain:

$$\boldsymbol{x}'\boldsymbol{x}f_{11} + \boldsymbol{x}'\boldsymbol{y}f_{12} + \boldsymbol{x}'f_{13} + \boldsymbol{y}'\boldsymbol{x}f_{21} + \boldsymbol{y}'\boldsymbol{y}f_{22} + \boldsymbol{y}'f_{23} + \boldsymbol{x}f_{31} + \boldsymbol{y}f_{32} + f_{33} = 0 \tag{2}$$

The *n* pair of points obtained by the matching method derive in *n* linear equations to estimate the fundamental matrix. However, one of the properties of the fundamental matrix, that must be necessarily fulfilled is that it is of rank 2. The solution obtained by the set of linear equations shown in equation (2), will be, in general, a matrix of rank 3. Therefore, the determinant of the obtained matrix must be forced to be zero once we have obtained a solution based on equation (2).

The minimum number of point matches we need to find the fundamental matrix is seven, as it will be explained now. Having a set of linear equations of the form A*f = 0, formed by seven equations as the one stated in (2), the solution is a two dimensional space of the form:

$$\boldsymbol{F} = \alpha \cdot \boldsymbol{F_1} + (1-\alpha) \cdot \boldsymbol{F_2} \tag{3}$$

Where α is a scalar variable and the matrices F1 and F2 are obtained as the matrices corresponding to the generators f_1 and f_2 of the right null-space of A. By forcing the determinant to be zero as explained before we obtain:

$$\det(\alpha \cdot \boldsymbol{F_1} + (1-\alpha) \cdot \boldsymbol{F_2}) = 0 \tag{4}$$

Since F_1 and F_2 are known, this leads to a cubic polynomial equation in α. When solving the equation, we can obtain one or three solutions (rejecting the complex ones), which leads, when substituting in equation (3), also to one or three solutions of the fundamental matrix.

If the number of points available is higher than seven, the so called 8-point algorithm can be used, being a simpler algorithm than the explained previously. In this case, the resolution of the linear equations leads to a unique solution by applying the equation used before and forcing the determinant of the fundamental matrix to be zero. However, before solving the linear equations it is necessary to apply a normalization to the input points, improving the stability of the result. The needed normalization is a translation and scaling of each image placing at the origin of the coordinates the centroid of the reference points so that the RMS distance of the points from the origin is equal to the square root of 2.

As explained, assuming we have a large amount of point matchings (more than 8), for estimating the fundamental matrix it is possible to use the 7-point algorithm or the 8-point algorithm. The application of each of the methods implies some advantages and disadvantages, however, the function used by MatLab to estimate the fundamental matrix is the 8-point algorithm, being its modification too complex and out of the scope of the project. Therefore, the fundamental matrix is obtained by applying the 8-point algorithm.

2.3.1.5. Relative position calculation

Once the fundamental matrix is known, we want to find the relative position of the camera in the second picture compared to the first one. This will be given by a rotation matrix (R), and a translation vector (t), which is unitary, according to the following expression:

$$\boldsymbol{E} = \boldsymbol{t} \cdot \boldsymbol{R} = \boldsymbol{K}^T \cdot \boldsymbol{F} \cdot \boldsymbol{K} \tag{5}$$

Where F is the fundamental matrix and E is the essential matrix, which is also a representation of the relation of the matched points but integrating the camera parameters. K is the intrinsic matrix, of the following form:

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Where f_x and f_y are the focal lengths of the camera in x and y respectively, x_0 and y_0 represent the offset of the principal point of the camera and s is the axis skew, which represents the skew coefficient between the x and y axis, which is normally zero.

The intrinsic matrix is given in the datasets that are being used, so the Essential matrix can be easily obtained. To obtain the translation vector and the rotation matrix, E must be decomposed by SVD, obtaining:

$$\boldsymbol{E} \approx \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T \tag{7}$$

Where U and V are orthogonal 3x3 matrices while D is a diagonal matrix of the form:

$$\boldsymbol{D} = \begin{pmatrix} \mathcal{E} & 0 & 0 \\ 0 & \mathcal{E} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{8}$$

Where $\mathcal{E}$ are the eigenvalues of E which must be two identical values and a null one. Assuming the following matrix to help solve the problem, which is later verified to be part of the result:

$$\boldsymbol{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

The rotation matrix and the translation vector are defined as:

$$\boldsymbol{R} = \boldsymbol{U} \cdot \boldsymbol{W}^{-1} \cdot \boldsymbol{V}^T \tag{10}$$

$$\boldsymbol{t} = \boldsymbol{U} \cdot \boldsymbol{W} \cdot \boldsymbol{D} \cdot \boldsymbol{U}^T \tag{11}$$

However, there are two possible solutions for the rotation matrix and another two for the translation vector, which implies four possible locations for the camera as it can be seen in the following image, where the points A and B represent the centres of each camera with its corresponding possible 2D projections (images) of a real world 3D point, represented as a big black point.

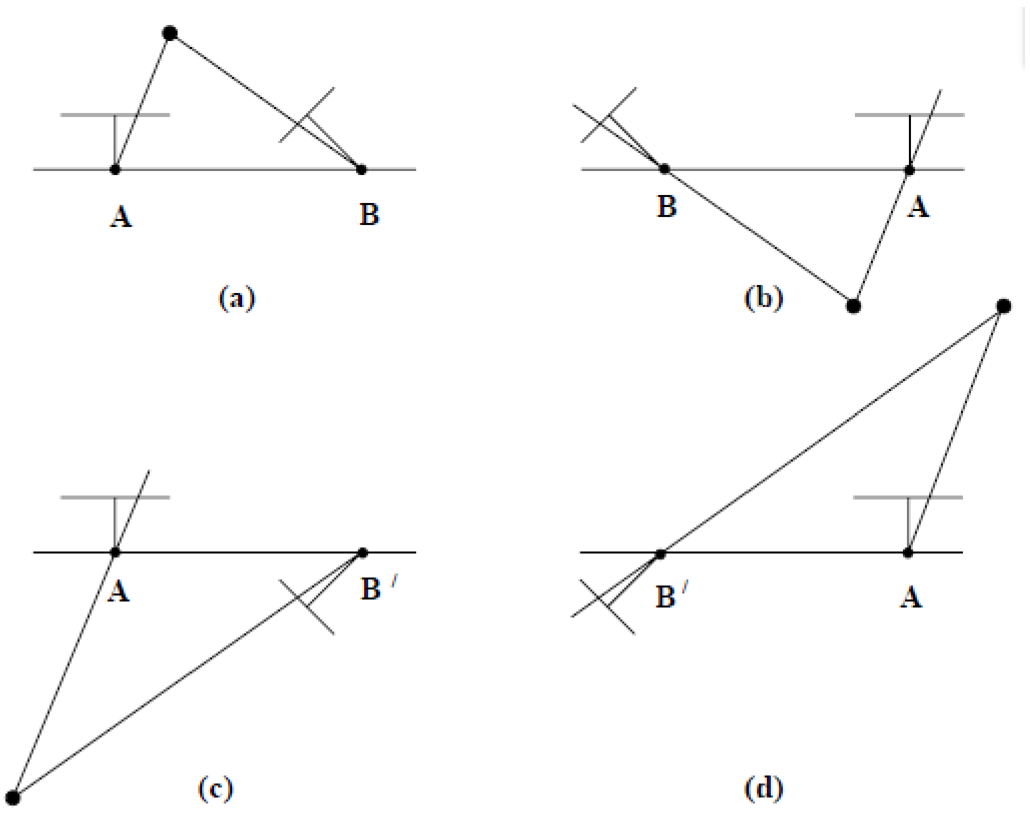


Figure 18: Possible camera positions

As it can be seen, the only possible solution is (a) as the points are in front of the cameras. This is checked by analysing the position of a single point with respect to both cameras and checking if it is positive. The rejection of the three other solutions implies also the rejection of some pair points that are found to be outliers with the new situation. Of course, it is important to check which is the percentage of rejected points as if it is high (more than 20 %), the fundamental matrix should be recalculated, as it means that it might have not been calculated correctly.

Once these parameters are obtained, it is just needed to add the information provided by the IMU, in this case, the velocity, to find how much the car is displaced per frame and multiply it by t so we obtain the absolute displacement of the car.

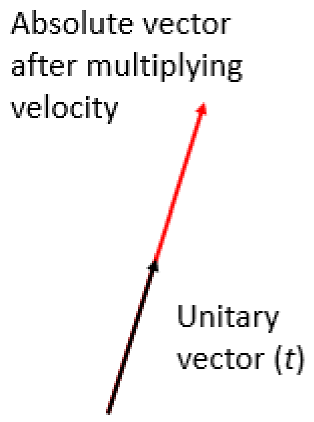
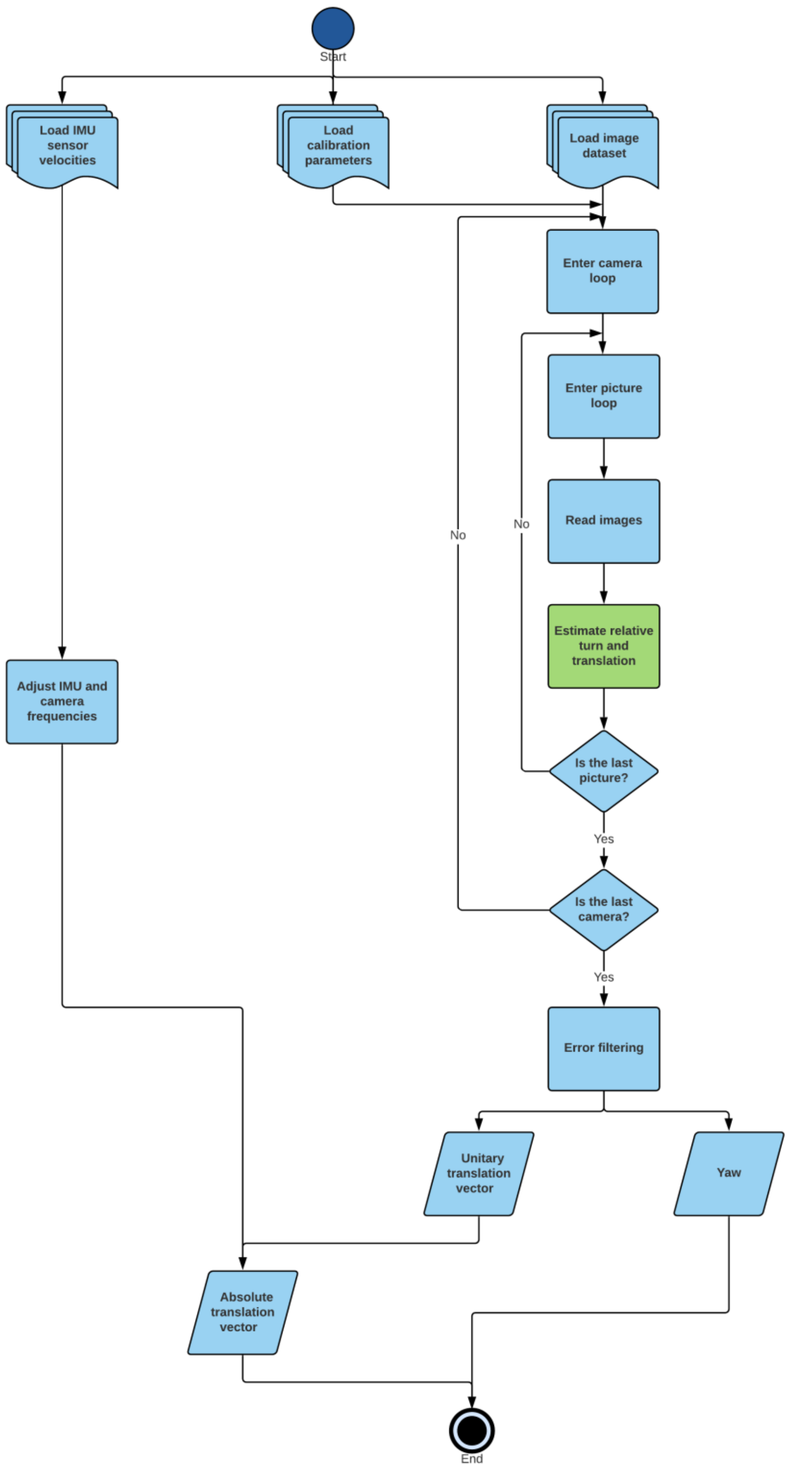


Figure 19: Multiplying the velocity to the direction vector

2.3.1.6. Functions

Once the algorithm has been explained from a theoretical point of view, the steps followed by each of the functions of the code are described below together with a diagram so it can be clearly understood. The cells in blue represent actions with no further development, while the cells in green represent actions defined by a subfunction that will be explained in a more detailed diagram later.

2.3.6.1.1. Main


Start
Load IMU sensor velocities
Load calibration parameters
Load image dataset
Enter camera loop
Enter picture loop
Read images
Estimate relative turn and translation
Is the last picture?
No
Yes
Is the last camera?
No
Yes
Adjust IMU and camera frequencies
Error filtering
Unitary translation vector
Yaw
Absolute translation vector
End

Graph 1: "Main" diagram

First, the image database to be studied is loaded by introducing the path where the images are located. Apart from it, the calibration parameters of each of the four cameras will be needed and also the velocities detected by the IMU, which will be used later.

The images are picked in pairs (1-2, 2-3, 3-4…) and introduced in the position estimator for each of the cameras. Afterwards, they go through an error filter that corrects calculations that could not be made because there was not enough data to estimate the fundamental matrix or the data used is found to be unreliable. Finally, we obtain a set of data corresponding to the yaw variation and another set corresponding to the unitary translation vector from frame to frame. To obtain an absolute translation vector we will have to make use of the IMU data.

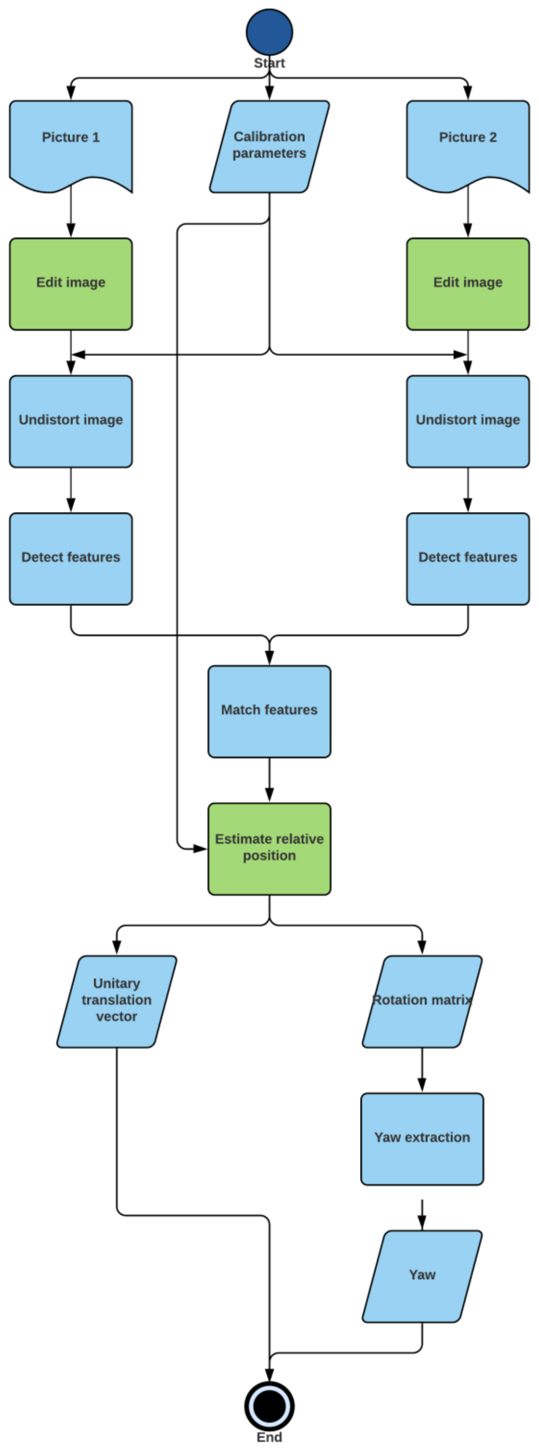
Before introducing the IMU data the obtained output goes through an error filter. The error filter is responsible of detecting the frames where the calculation could not be done or the points where the calculation was wrong. If the calculation could not be done, the filter will find a zero and will substitute it by the average value obtained from the two closest values to it (the one before and the one after in general). To determine if there is an error, the maximum possible value for the yaw variation has been selected by studying the changes in the data from frame to frame and detecting visually, possible anomalies. After the study, it is decided that the yaw variation from one frame to the following one whose value is higher than 0.33 degrees will be considered as a wrong calculation, carrying out the same process as before for its correction.

The IMU has a frequency ten times higher than the camera, therefore, it gives ten times more measures. To adjust the frequencies, the average value of the ten velocities corresponding to each frame is calculated. Then, the distance travelled per frame is calculated as:

$$displacement\ (m) = \frac{1}{10}(s) * velocity\ \left(\frac{m}{s}\right) \tag{12}$$

The ten corresponds to the frequency of the camera. Finally, the obtained distance is multiplied to the unitary translation vector to make it absolute.

2.3.1.6.2. Position estimator



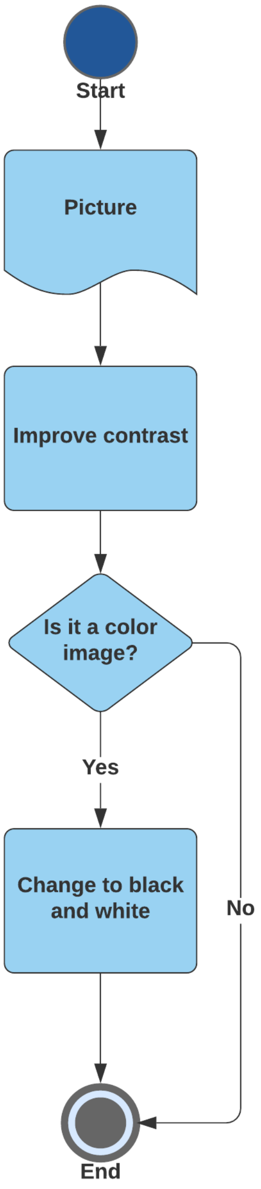
Graph 2: "Position estimator" diagram

The position estimator function is responsible of obtaining the unitary vector, showing the direction of movement, and the yaw variation of the camera between the frames by receiving two consecutive images and the corresponding calibration parameters of the camera.

First, the images are edited to fit the algorithm, as explained later, and then corrected for lens distortion according to the camera calibration parameters. Then, the features of each of the images are detected and extracted so they are matched according to the algorithm explained in the previous section.

Once the matching process is done, the matched points are introduced in a function to estimate the desired parameters. However, instead of obtaining directly the yaw variation from the function a rotation matrix is calculated, from which the yaw variation can be easily extracted transforming the matrix into Euler angles.

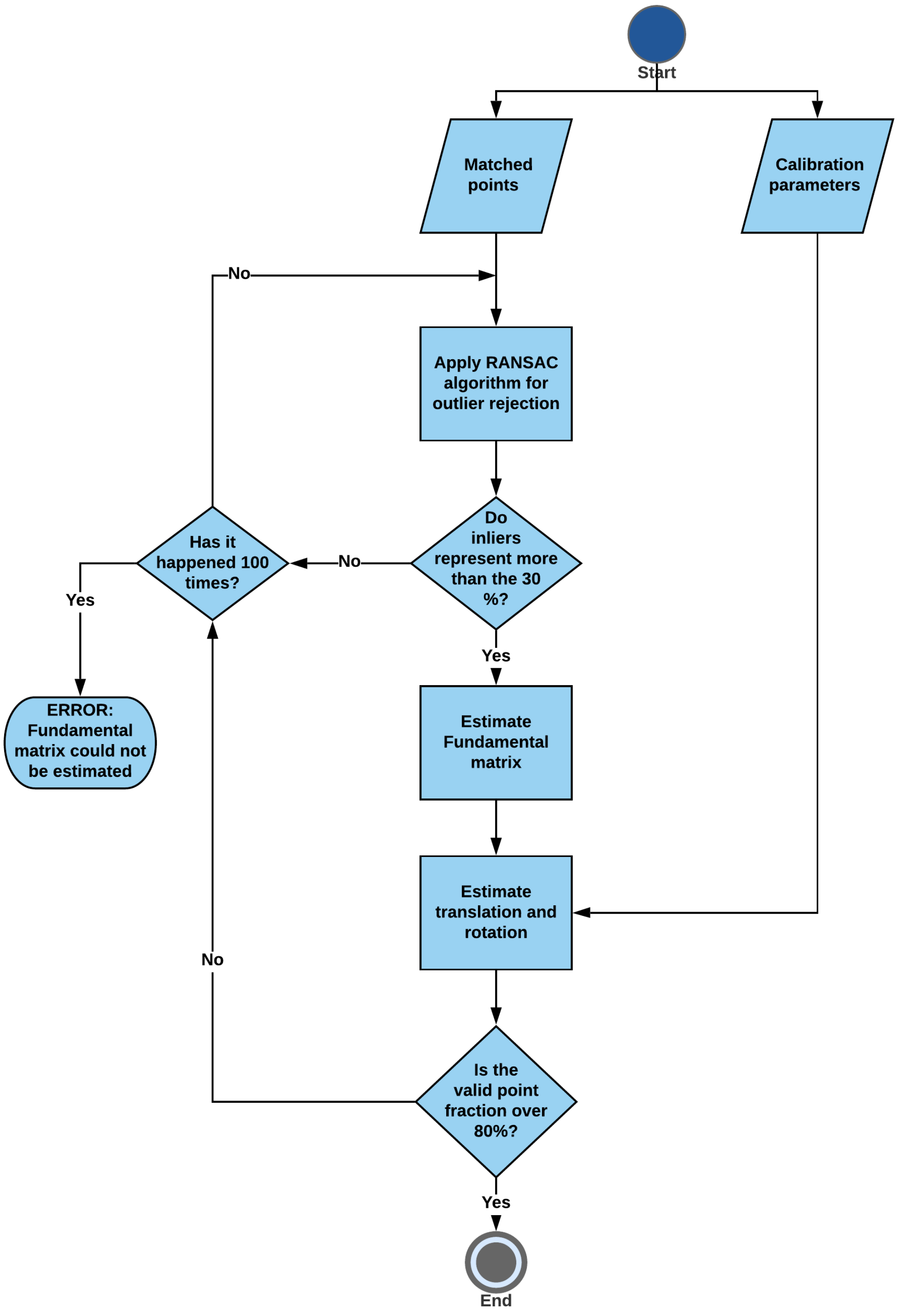
2.3.1.6.3. Edit image



Graph 3: "Edit image" diagram

The edit image function is responsible of improving the contrast of each image for better feature detection and, if the picture is a color image, converting it to grey to enable the feature detection.

2.3.1.6.4. Estimate relative position


Start
Matched points
Calibration parameters
No
Apply RANSAC algorithm for outlier rejection
Do inliers represent more than the 30 %?
No
Has it happened 100 times?
Yes
ERROR: Fundamental matrix could not be estimated
Yes
Estimate Fundamental matrix
Estimate translation and rotation
No
Is the valid point fraction over 80%?
Yes
End

Graph 4: "Estimate relative position" diagram

The estimate relative position function is responsible of obtaining the rotation matrix and unitary translation vector corresponding to the camera movement between the frames. It is important to note that in this step only the matched points together with the camera calibration parameters are introduced as inputs, not needing the images any more. This means that once the pairs of points are matched from the images, the following steps are based on an algebraic problem.

The first step of the function is to apply the RANSAC algorithm to reject the outliers found in the matching process. It is then checked if there are, at least, a 30 % of pair of points that have not been rejected. If the inlier amount is lower than the 30 percent, it is considered that the picked points for the RANSAC algorithm are not good choices, so the function is restarted.

Assuming that enough inliers have been detected we advance to the next step, where based on the inliers obtained previously, the fundamental matrix is estimated according to the process explained in 2.3.1.4. With the fundamental matrix and the camera parameters we can then obtain the essential matrix and finally the rotation and direction of translation of the camera.

However, at this point it must be remembered that there are two possible solutions for the rotation matrix and another two for the translation vector. To obtain the correct solution is easy, as explained before, but we have to take into account that the rejection of solutions will also imply the rejection of some of the used inliers for the calculation. If more than twenty percent of the points are rejected the calculation is, once again, considered invalid and we go back to the start and recalculate.

If the function fails, in any of the two conditions a total number of a hundred times (which is considered to be an enough number of iterations to obtain proper results), it is considered that the provided information (the matched points) are not correct, which in general, indicates a problem in the images used for the calculation.

2.3.1.7. Results and validation

As explained before, to validate the position estimator two different kinds of validations will be carried out. First, a visual validation will be done for validating the yaw results using just the geometry of the road and then, by using a different dataset, the results obtained will be compared with the ones obtained with different sensors, to see how close the measurements are and to see if they are logical.

2.3.1.7.1. Visual validation

The dataset used consists of 425 images (42.5 seconds) describing a car in a highway. The car exits the highway from the right, describing a long right turn of 270 degrees with a short straight part in the middle to enter again a second highway that crosses the first one perpendicularly as it can be seen in the shown frames.

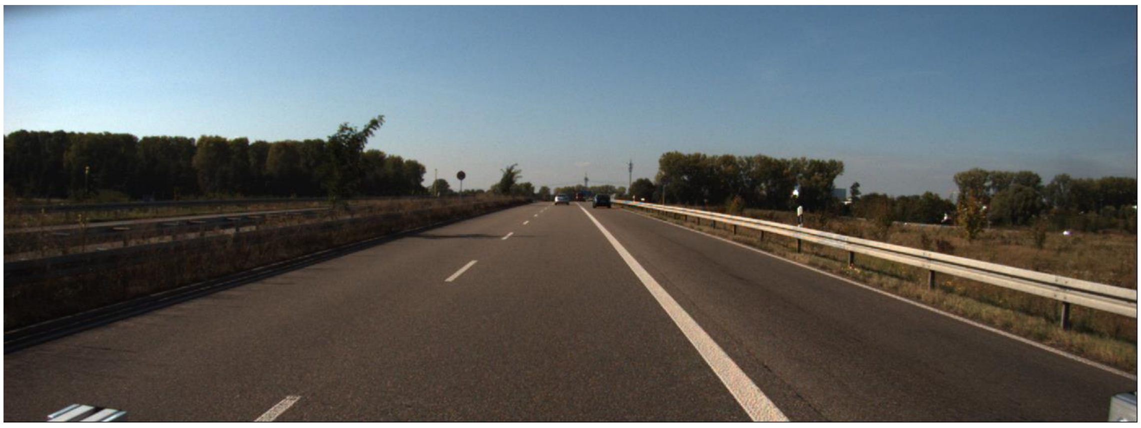


Figure 20: Car in highway

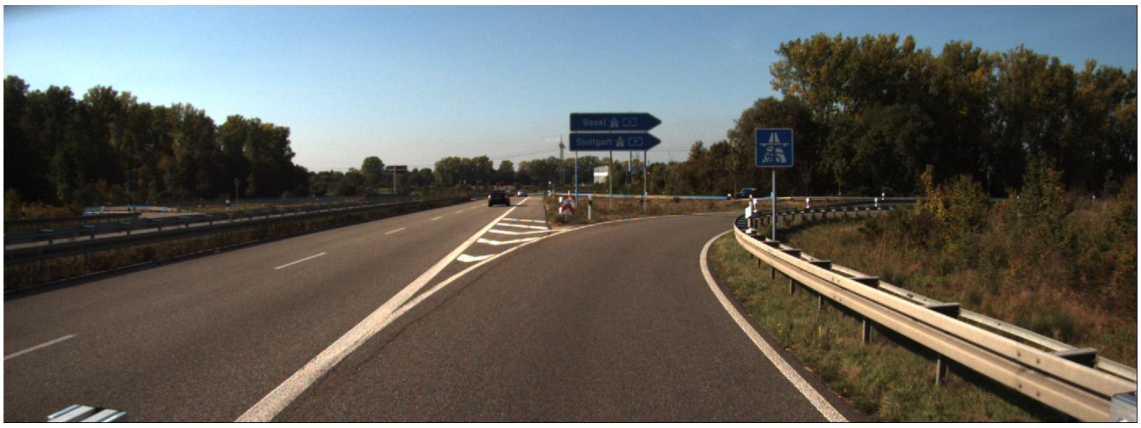


Figure 21: Car exiting highway

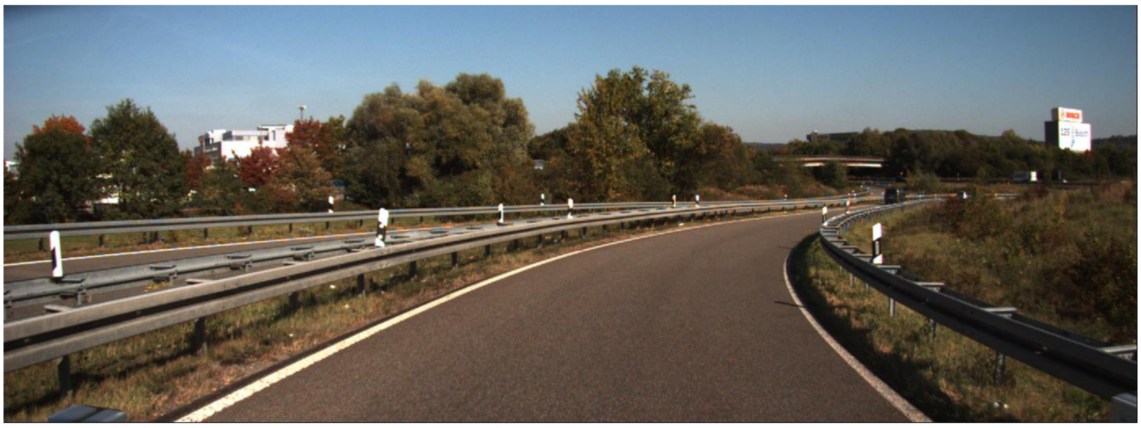


Figure 22: Car turning right

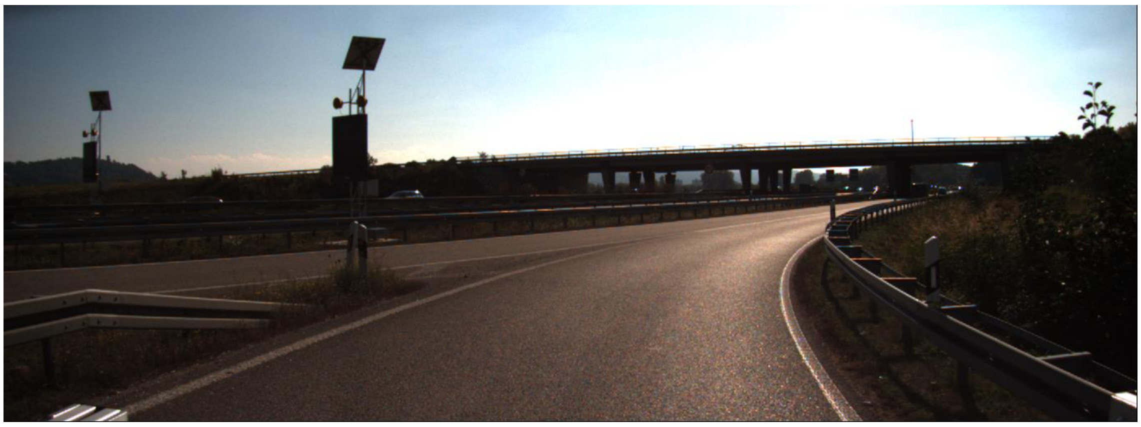


Figure 23: Car entering highway

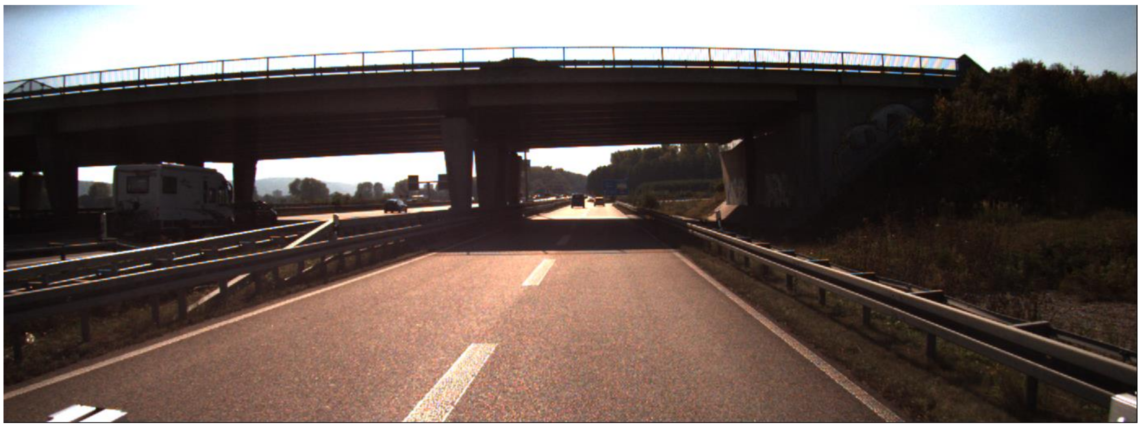
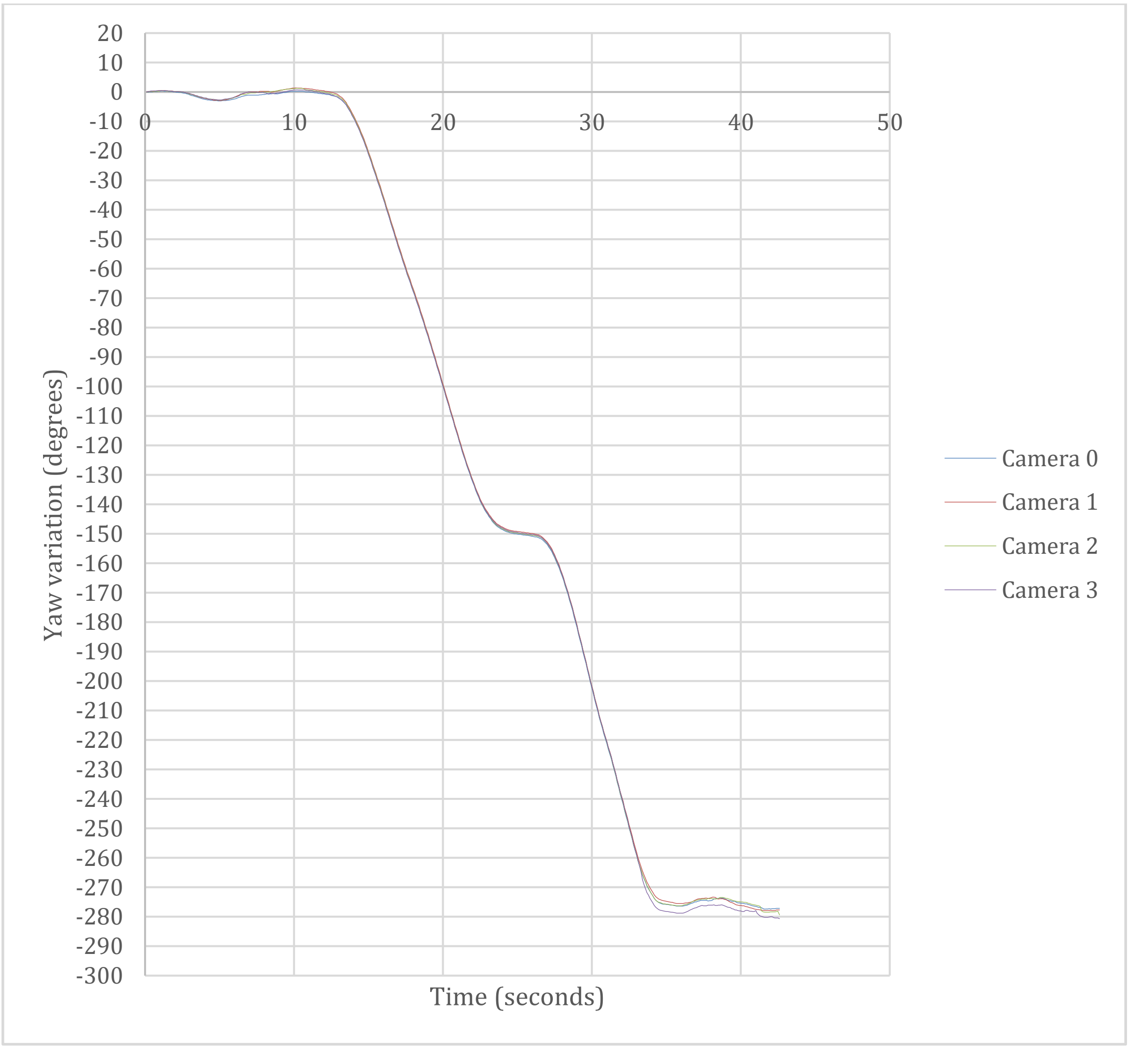


Figure 24: Car about to cross perpendicularly the first road section

By doing a simple visual inspection, it can clearly be seen that the total yaw of the car across the way should be 270 degrees approximately, this is, three fourths of a circle.

The yaw is calculated from frame to frame and then all the obtained measurements are added to obtain which is the total turn described by the car in the chosen dataset. The calculation is done for the four cameras available in the car, converting, as explained, the color images of the two color cameras to grey. It is important to take into account that the results obtained by the color cameras are expected to be worse than the ones obtained with the grey ones because the images taken by them have less contrast, being this a key factor in the feature detection process. The results obtained by each of the cameras regarding the yaw variation are plotted in the following graph, where the camera 0 represents the left grey camera, the camera 1 represents the right grey camera, the camera 2 represents the left color camera and the camera 3 represents the right color camera.



Graph 5: Yaw measurement results

The total yaw calculated by each of the cameras 0, 1, 2 and 3 are -277.14 degrees, -277.69 degrees, -279.39 degrees and -280.67 degrees respectively. We can see that the results obtained are very close to each other and also close to the 270 degrees approximate result that we expected. However, it is important to be aware that if the calculation is repeated with exactly the same input data, the results will vary due to the random selection of points of the RANSAC algorithm. Anyway, the total value obtained should be close to the obtained ones in this experiment.

By looking at the graph, it is interesting to take a look at the smooth variation between the start of the dataset and the tenth second and between the seconds 35 and the end, representing in both cases a change of lane. It can also be clearly seen the short straight part done when the car was exiting the highway, which demonstrates that the algorithm is working well.

2.3.1.7.2. Validation by comparison

Once it has been checked that the system is working properly and giving proper results, we can compare the results with the measurements obtained by the GPS to quantify the accuracy of the system.

In this case, the dataset used contains a set of 666 images in a residential environment. The car is finishing a left turn before facing a long straight. Then it turns 90 degrees to the right, continuing straight and finishing with a left turn of, again, 90 degrees.

Figure 25: Car going straight

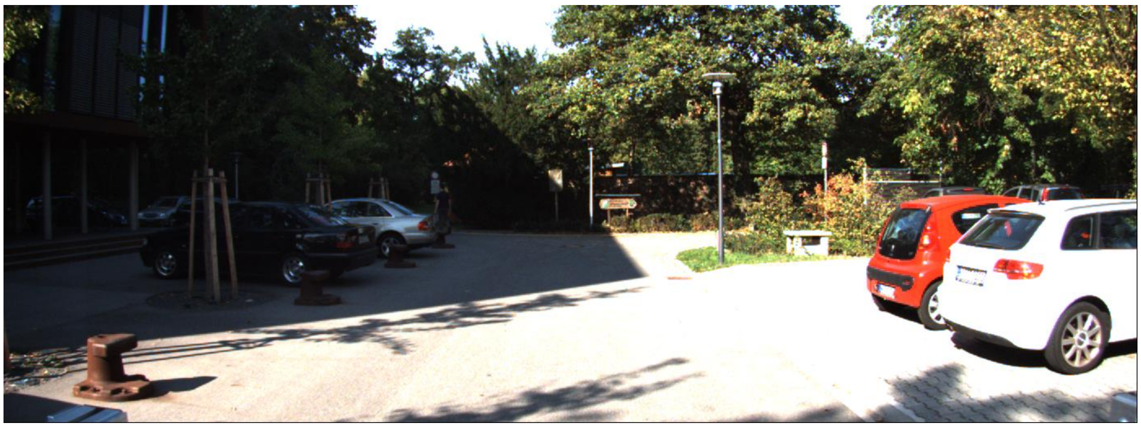


Figure 26: Car going straight

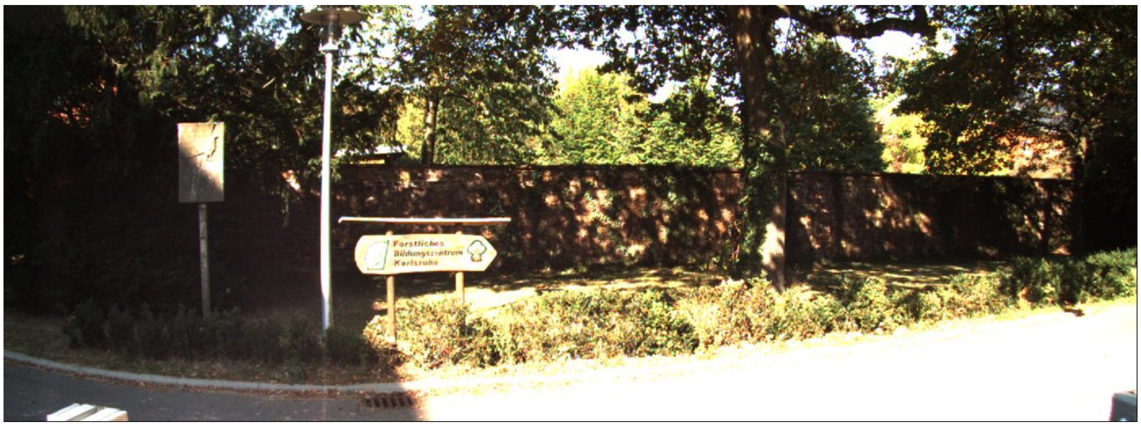


Figure 27: Car turning right

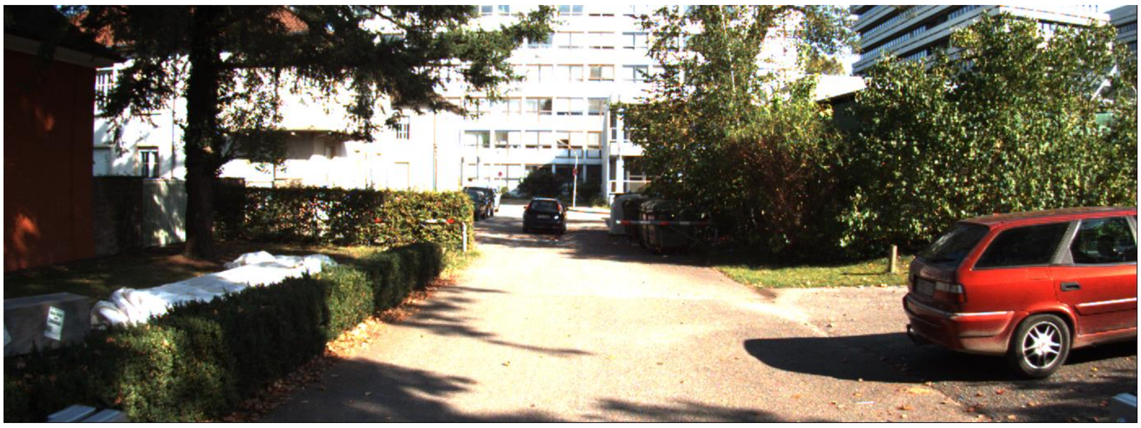
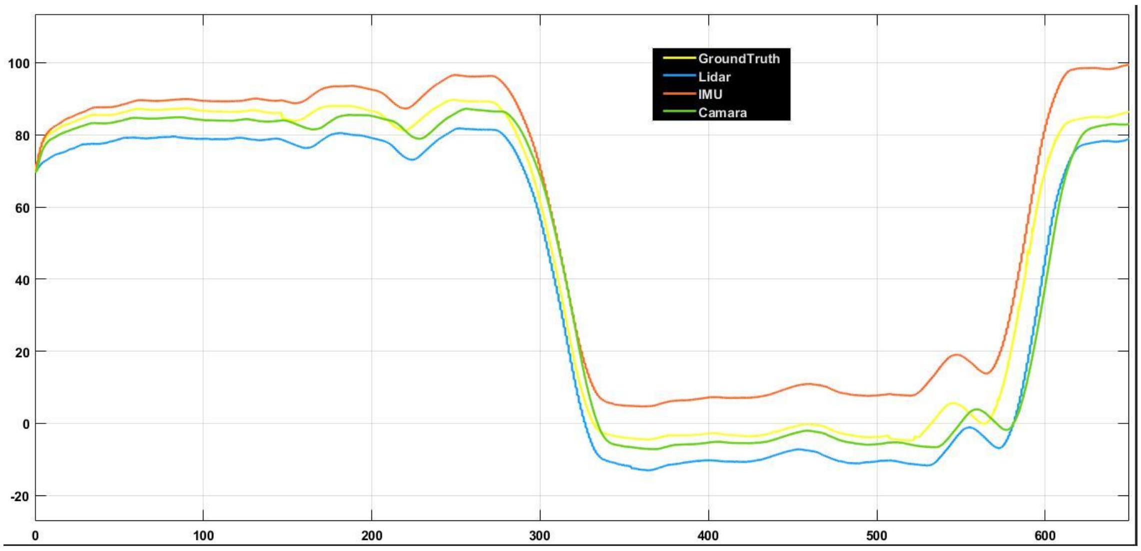


Figure 28: Car going straight

Figure 29: Car about to turn left

The yaw has been calculated with different sensors and plotted in the same graph as it can be seen in the following picture:

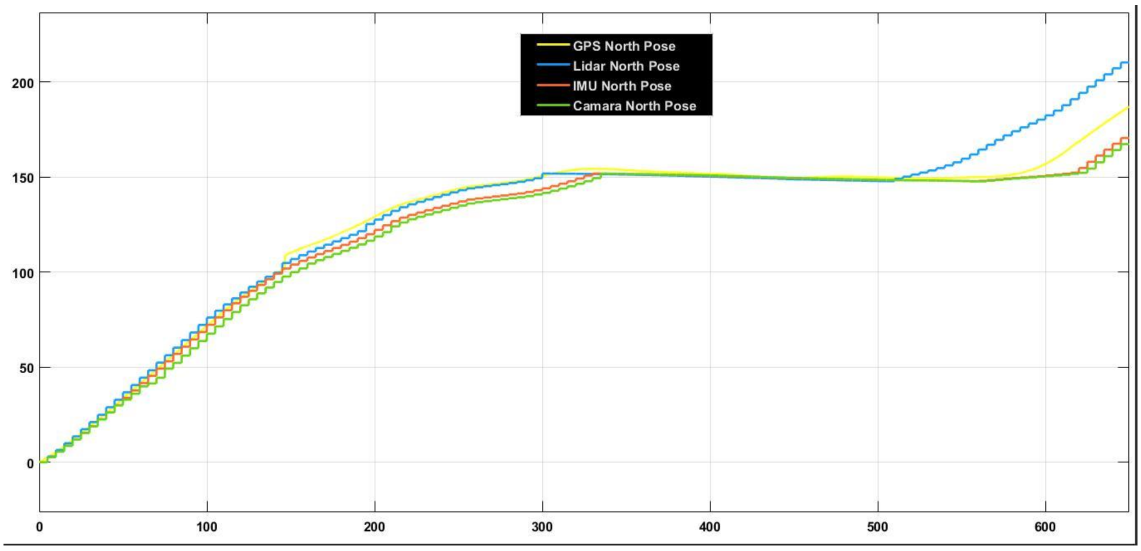


Graph 6: Yaw results with different sensors

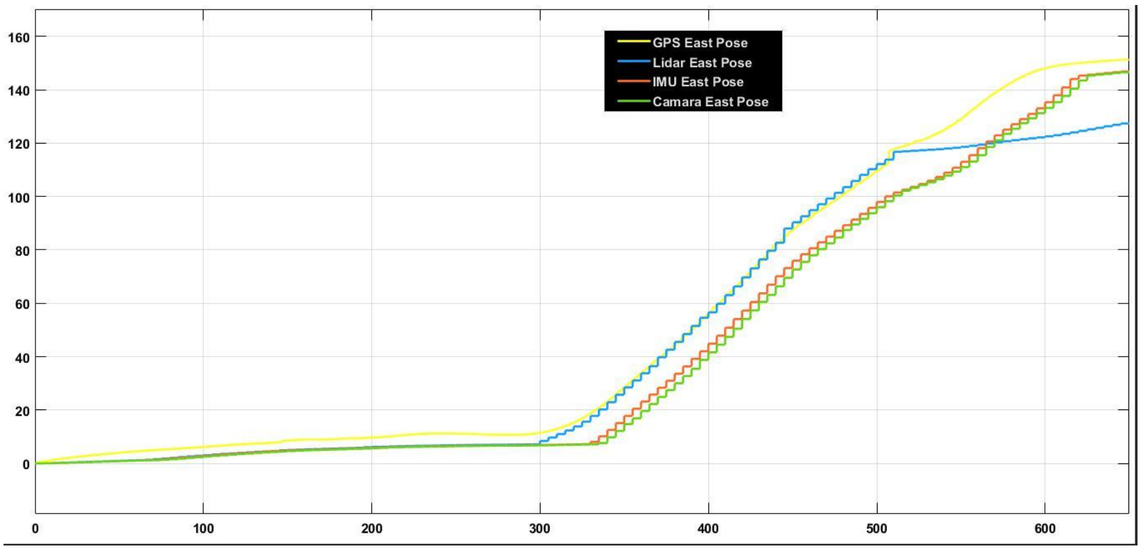
The line in red represents the measurements obtained with the IMU, the yellow line represents the measurements obtained with the GPS, the green line represents the measurements obtained with the camera and finally, the blue line represents the measurements obtained with the LiDAR. The two 90 degrees curves are clearly detected as it can be seen in the parts with high slope of the graph, which indicate a yaw variation.

As it can be seen, the results obtained with the camera are logical compared to other sensors and stays closer to the GPS measurements than any of the other two. The GPS is assumed to be the most accurate sensor of the presented ones. However, this is not always true, so it is impossible for us to measure with a percentage the accuracy of the algorithm or show its error.

The position results are compared in two different graphs. One of them shows the displacement to the north against time and the other one shows the displacement to the east against time.



Graph 7: Displacement to north with different sensors



Graph 8: Displacement to east with different sensors

The sensors are represented with the same colors as in the yaw graph. It can be clearly seen that the IMU and camera results are almost the same in all the graph, implying that the effect of the unitary vector obtained in the matching process is minimum compared to the effect generated by the IMU velocity magnitudes, being an interesting point to work on in future projects. However, as the results for the camera define the two curves in the set (which is, mainly, responsibility of the unitary vectors), we can state that the results obtained are logical.

To summarize, it can be conclude that the algorithm is quite good estimating the yaw variation of the car, but that the limitation of needing a second sensor to calculate the displacement from frame to frame derives in worse results in this parts, or at least not useful results for the sensor fusion due to the dependence of the IMU.

2.3.2. ROAD CLASSIFIER

The position estimator is a great tool to analyse the position of the car in short distances, however, as time goes on, the error that the estimator might have keeps increasing. For that reason, it is interesting to relocate the car somehow from time to time. Assuming that the maps are known, if we could detect that the car has reached a crossroad or a curve, we could replace its position for the location of the instance detected.

The first task is to separate the road from the rest of the image. The best way to do this, as explained in section 1.6.3, is by training a segmentation CNN, which will allow us to classify the pixels of the image in two categories: “Road” and “Rest”. Normally, what most time takes when developing a CNN is obtaining proper images and labelling them. However, we can take advantage of the training set provided in the KITTI database and skip this step.

The KITTI training images make a distinction between the road corresponding to the sense the car is going and the counter sense road, therefore, they have three labels. As we are not interested in detecting the counter sense, we have to preprocess the labels before training and reduce the categories to two.

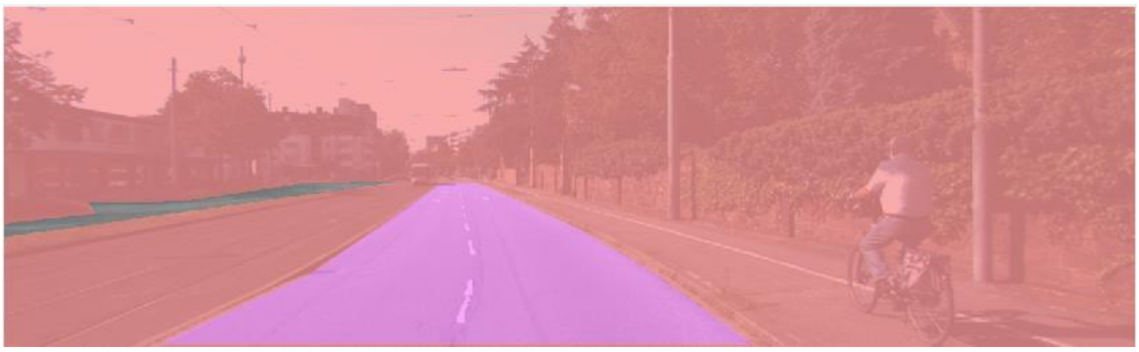


Figure 30: KITTI labeled image before preprocessing

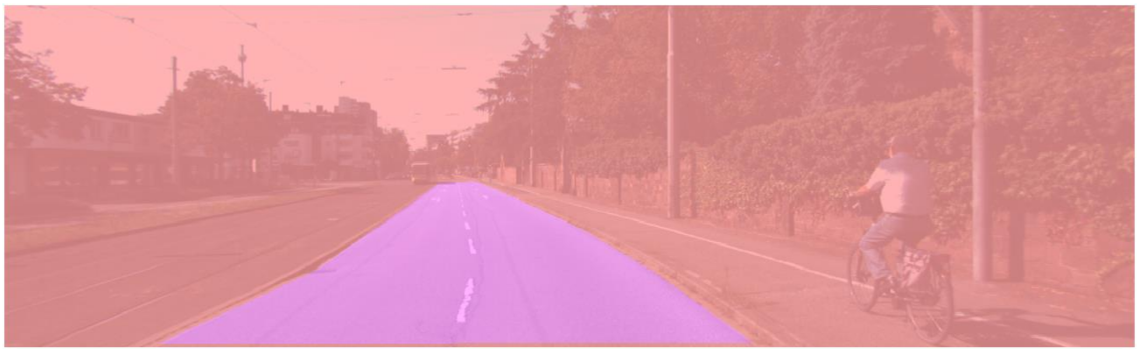


Figure 31: KITTI labeled image after preprocessing

Instead of creating our own CNN for segmenting the images, we create a segmentation network based on the pretrained network Resnet 18, using the transfer learning technique mentioned in 1.5.3 and appearing in [12]. This reduces the training time and complexity. We choose Resnet18 as it can be easily modified to a segmentation network and due to its good accuracy-prediction speed ratio relation, as it can be seen in the following image.

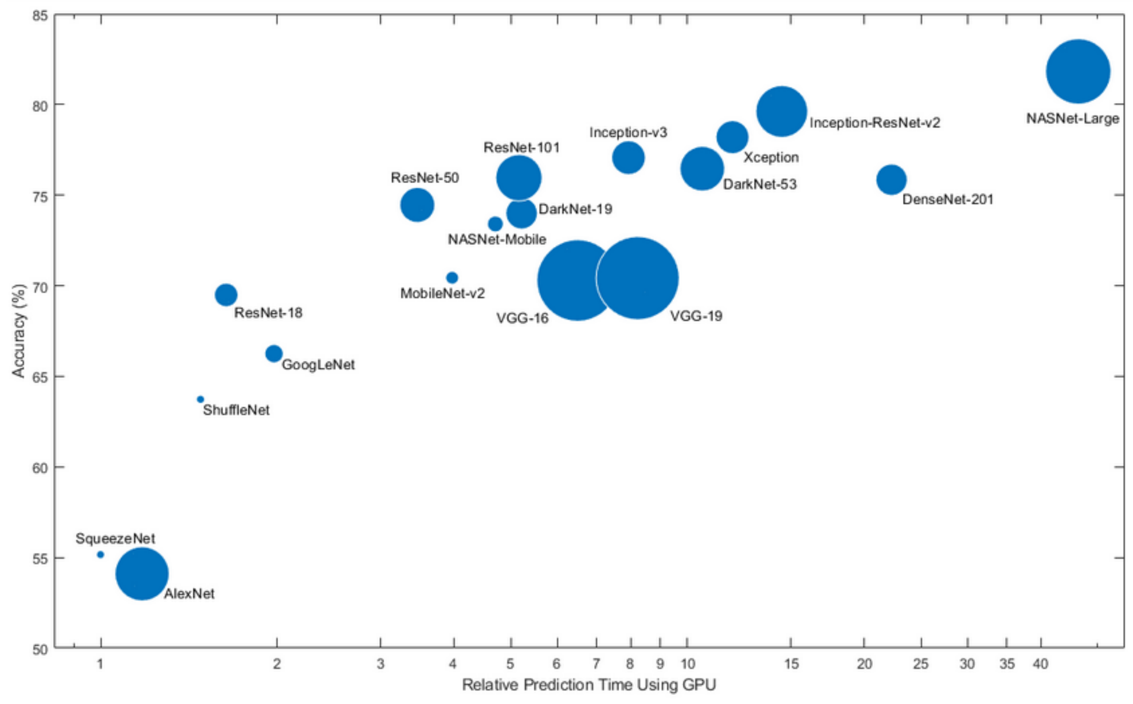


Figure 32: Pretrained CNNs accuracy vs prediction speed

As in the KITTI database labels are given to train also a lane segmentation network, we take advantage of it and create this network as well. The training code used for the road detector can be reused for the lane detector, so it does not suppose an increase of work. However, this CNN was not found useful in this project to fulfil its objectives, so it will not be used in further calculations. Nevertheless, its development might be useful in different tasks such as finding what lane of the road the car is at, by overlapping the two CNNs.

For the road segmentation CNN, 289 training images are used of three different types of roads: urban unmarked, urban marked and urban multiple marked. For the lane segmentation CNN, 95 training images of urban marked roads are used. Both CNNs are tested with a set of 290 images with successful results.

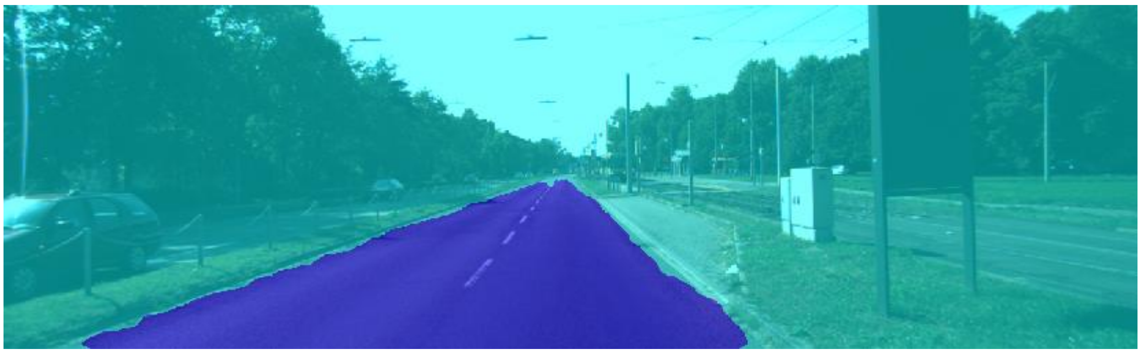


Figure 33: Road segmentation CNN

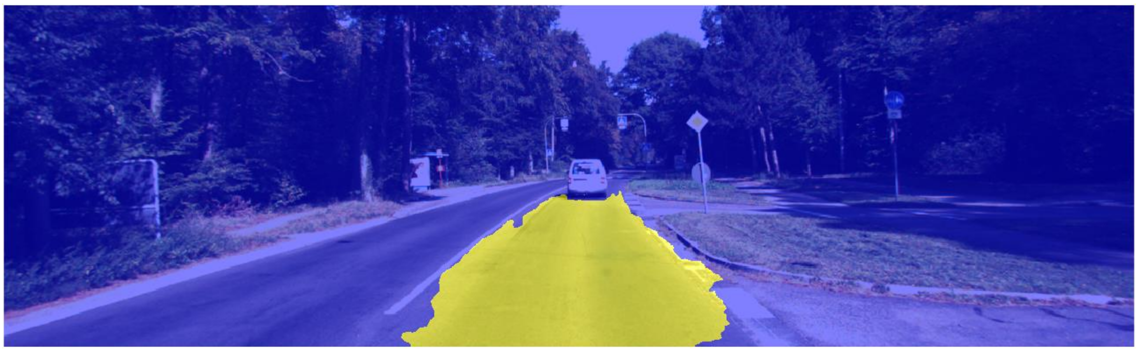


Figure 34: Lane segmentation CNN

Once we are able to divide the road from the rest of the image, the detection is postprocessed with the objective of eliminating the noise that appears in some of the images and to obtain numerical data about the road shape that will be then used to train the road classifier.

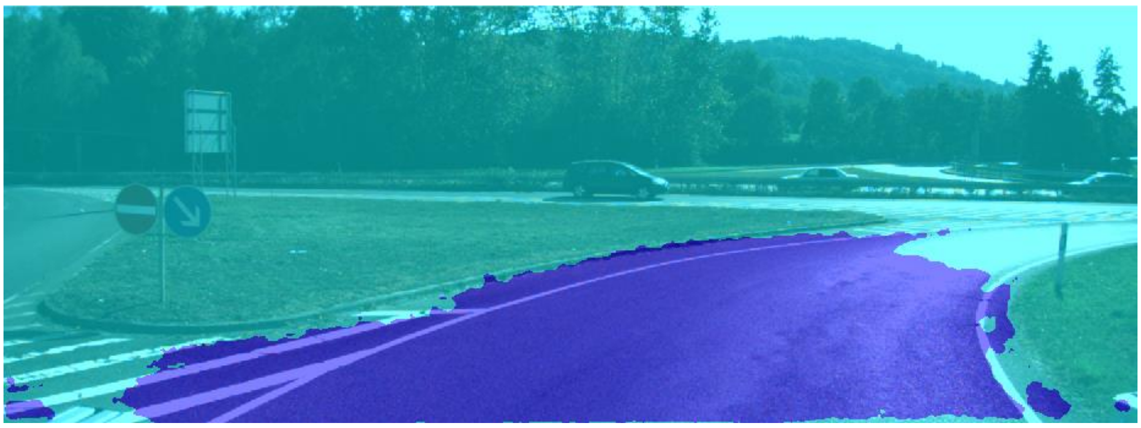


Figure 35: Noise corrupted detection

The first step is to cut the lower part of the image, where the network does not detect well the first pixels of road. Then, an auxiliar image is created where the road detection appears in white while the rest appears black. After that, we smooth the edge where black and white colours collide with a gauss filter and delete the pixel groups of the same color that are smaller than 15000. This number has been selected according to the maximum pixels that can gather together as noise and introducing a security margin of 5000 pixels. In general, the group of pixels belonging to the road detection without noise is far over this number.

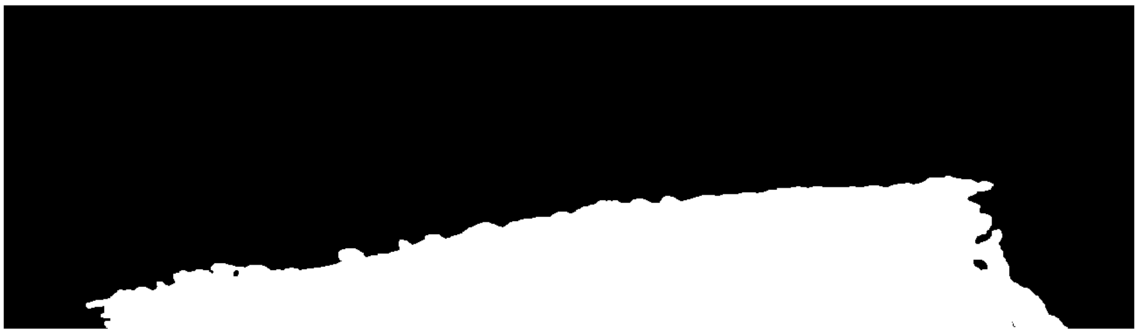


Figure 36: Noise filtered auxiliar image

As it can be seen in the image, the small groups of pixels detected as "Road" have disappeared as they represent a smaller number than the specified. It is important to make these two steps as they could lead to big errors when extracting data from the detection.

The set of images needed for the classifier is not as easy to obtain as the one used for the segmentation network because, in this case, any image where a road appears is not valid. In general, most of the images correspond to straights, while left curves are the less common ones. However, by flipping the images corresponding to right curves, which are more common, we can obtain the same number of left ones. By applying the flipping method we finally obtain 32 crossroads, 33 left curves, 33 right curves and 74 straights.

The option of training another CNN for the classification has been thought, which, in fact, would be the most robust option. However, as mentioned in 1.6.3, the images available for training are too few for a CNN to reach a proper accuracy in detection. That is the reason why instead of a CNN it is decided to make a simpler classifier by using the Classification Learner app of MatLab.

The Classification Learner app allows us to train different models with the data we import. Unlike a CNN training, where we could use the merely postprocess image as training, for training the different models in the app we need to give numerical data. Therefore, the next step is to extract characteristic points from the auxiliar image to obtain data for the classifier. The parameters we extract and the procedures followed to obtain them are described here:

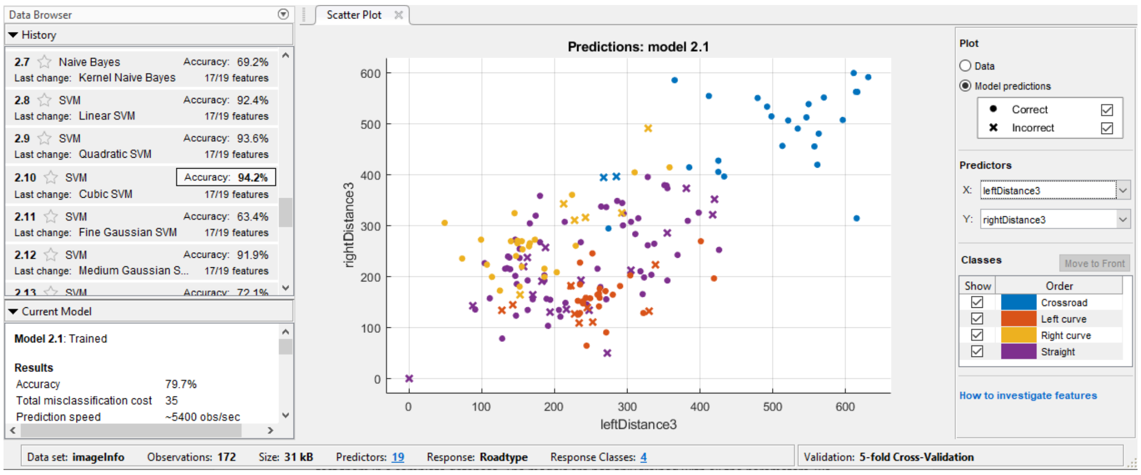
a) Parameter 1: Highest row where a white pixel appears.
b) Parameter 2: Middle point of road in the lowest row.

For obtaining the following parameters, we need to detect the pixels that correspond to each of the sides of the road. Then, these points are approximated with a straight line.

c) Parameter 3: Angle defined by the straight line that best fits the pixels on the left side of the road.
d) Parameter 4: Angle defined by the straight line that best fits the pixels on the right side of the road.

After that, a vertical line is defined that goes through parameter 2. In each row the distance from this line to the pixels that are in the sides of the road is calculated.

e) Parameter 5: Average value of distances from vertical line to pixels in the road left side.
f) Parameters 6, 7, 8, 9 and 10: Distance from vertical line to five equally spaced points in the left side.
g) Parameter 11: Average value of distances from vertical line to pixels in the road right side.
h) Parameters 12, 13, 14, 15 and 16: Distance from vertical line to five equally spaced points in the right side.
i) Parameter 17: Number of pixels detected that belong to the road.
j) Parameter 18: Parameter 5 divided by parameter 1.
k) Parameter 19: Parameter 11 divided by parameter 1.



2.7 Naive Bayes — Accuracy: 69.2%
Last change: Kernel Naive Bayes — 17/19 features
2.8 SVM — Accuracy: 92.4%
Last change: Linear SVM — 17/19 features
2.9 SVM — Accuracy: 93.6%
Last change: Quadratic SVM — 17/19 features
2.10 SVM — Accuracy: **94.2%**
Last change: Cubic SVM — 17/19 features
2.11 SVM — Accuracy: 63.4%
Last change: Fine Gaussian SVM — 17/19 features
2.12 SVM — Accuracy: 91.9%
Last change: Medium Gaussian S... — 17/19 features
2.13 SVM — Accuracy: 72.1%

Current Model

Model 2.1: Trained

Results

Accuracy	79.7%
Total misclassification cost	35
Prediction speed	~5400 obs/sec

Data set: **imageInfo** Observations: **172** Size: **31 kB** Predictors: 19 Response: **Roadtype** Response Classes: 4 Validation: **5-fold Cross-Validation**

Figure 37: Classification learner app

Due to the little time it takes to study all the possible models, it is possible to see which is the accuracy of each one and select the model based on that. Among all the classifiers the most promising types are the SVM (Support Vector Machine) and KNN (Nearest Neighbours Classifiers).

As it is not clear by analysing the theory which of the two models or its corresponding submodels (fine, quadratic, cubic) will fit better, we extract all the models with a relative high accuracy and test them in a complete database. The models are not only trained with all the parameters, we also try to remove some of them, as they could be leading to wrong results. The models extracted are the following:

- **Model 1:**
  - **Type of model:** Cubic SVM
  - **Used parameters:** All
  - **Accuracy:** 94.2 %
- **Model 2:**
  - **Type of model:** Fine KNN
  - **Used parameters:** All
  - **Accuracy:** 94.2 %
- **Model 3:**
  - **Type of model:** Cubic SVM
  - **Used parameters**: All but 3 and 4
  - **Accuracy:** 94.2 %
- **Model 4:**
  - **Type of model:** Cubic SVM
  - **Used parameters:** All but 3,4,5 and 11
  - **Accuracy:** 94.2 %
- **Model 5:**
  - **Type of model:** Fine KNN
  - **Used parameters:** All but 3, 4, 17, 18 and 19
  - **Accuracy:** 94.8 %

Apart from studying just the accuracy of the models, it is interesting to take a look at the confusion matrix of each of the models, which indicates with what category has the model mismatched an example in the validation process.

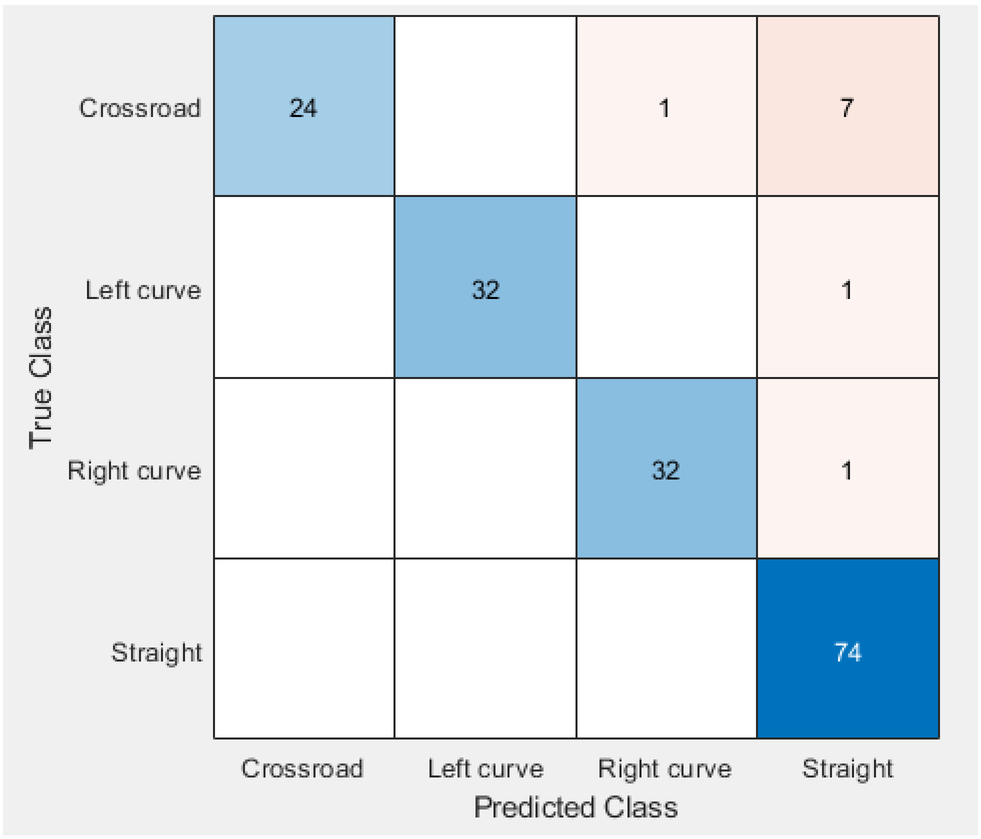


Figure 38: Confusion matrix for models 3 and 4

The confusion matrices of the five models have a similar shape, mismatching, in general, crossroads with straights. This fact, is probably due to the less accurate detection in the parts of the road appearing in the sides of the image and we can deduce in advance that most of the wrong detections in the validation process will occur when detecting crossroads.

To increase the robustness, a sliding window is introduced that takes into account the 4 previous predictions apart from the predicted one. If three of the five predictions are the same, that is the final result. If there is not a road type that appears three times out of five, the initial guess is maintained and the past guesses are not taken into account.

The models will be tested in the first dataset used for the validation of the position estimator and a new dataset describing a residential environment. In the dataset the car goes straight until it finds a crossroad, where it turns left. Then, it continues straight until finding another crossroad where it is stopped by a traffic light.

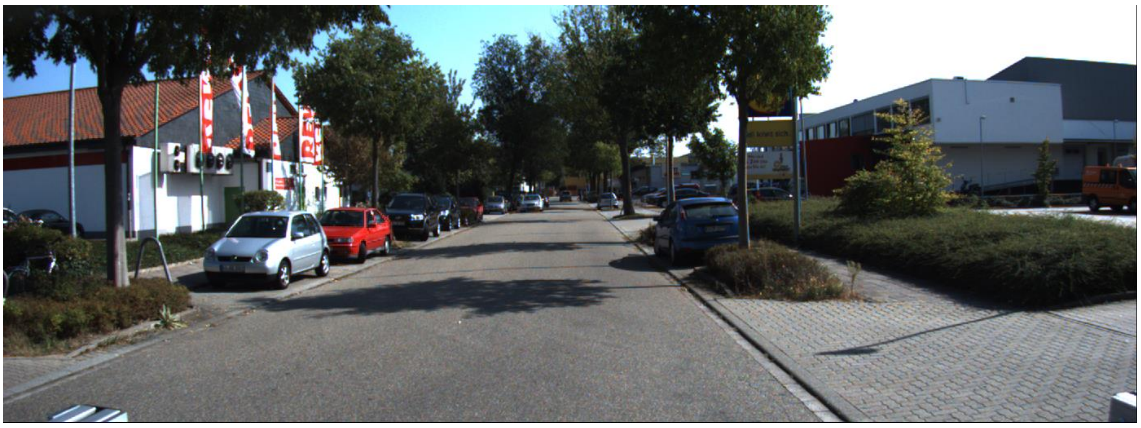


Figure 39: Car going straight

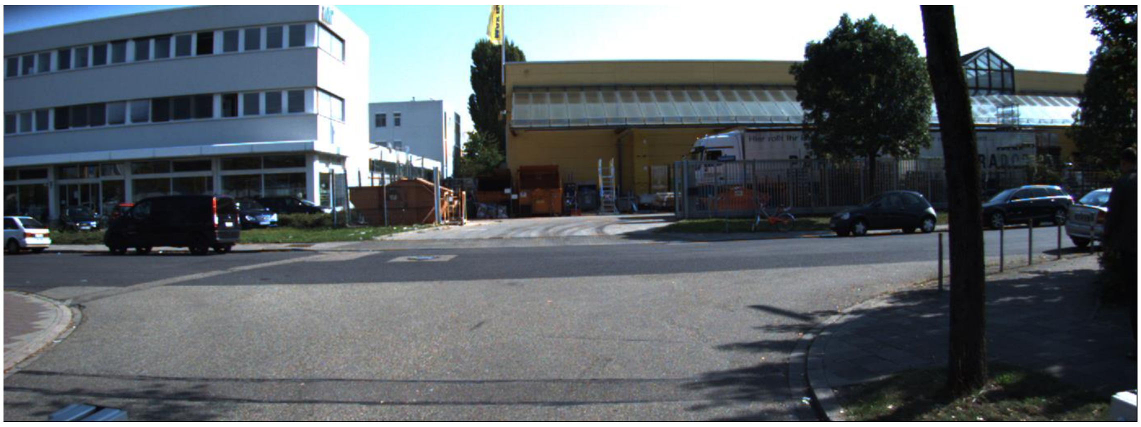


Figure 40: Car reaching crossroad

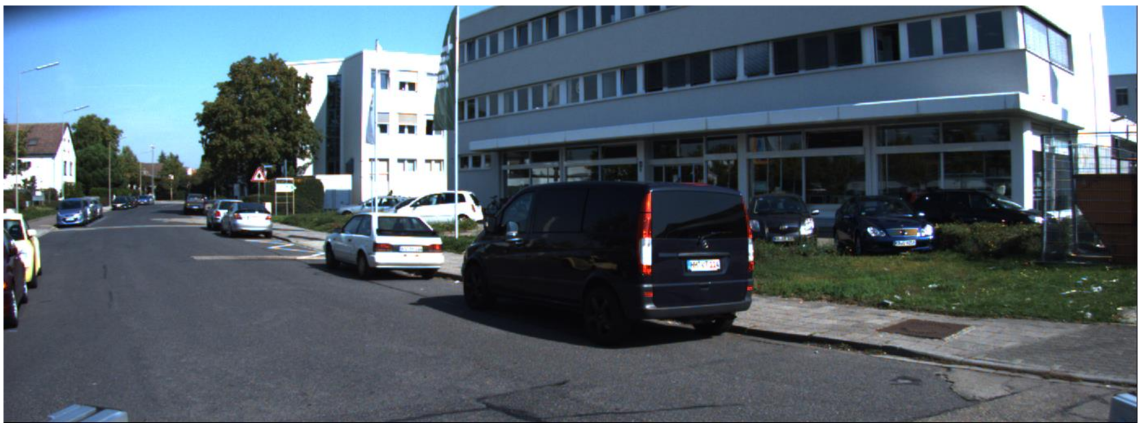


Figure 41: Car turning left

Figure 42: Car going straight

Figure 43: Car stopping at traffic light

The results hit percentage obtained by each of the models in each dataset can be seen in the following table:

Dataset	Classifier	Efectiveness
Highway	Model 1	91,63%
	Model 2	88,67%
	Model 3	91,13%
	Model 4	91,13%
	Model 5	87,93%
Residential	Model 1	93,18%
	Model 2	94,35%
	Model 3	82,82%
	Model 4	84,00%
	Model 5	93,65%

Table 2: Road classifier hit percentages

As it can be seen, the KNN models (2 and 5) show a good result in residential areas while in highways, they are clearly beaten by the SVM. By looking at the models that have considered all the extracted parameters from the images (1 and 2), being one an SVM and the other one a KNN, it can be said that considering as much parameters as possible leads to better results. Finally, the clear best model in overall is model 1, being the only model having an accuracy over the 90 % in both environments.

Even if the results seem high, it has to be taken into account that most of the images correspond to straights, which have already shown very good results in the confusion matrices presented before. Apart from that, the road classifier shows some limitations, being this the reason why the second dataset used for validating the position estimator is not used for validating the classifier.

The first limitation, is the disappearance of the road from the images in some of the turns of the car. If the road does not appear in the image, there is no way a guess can be made as the segmentation network does not give any output, we can only consider that we are not going straight. In the figure 27 it can be seen that the road only appears in the lower part of the image, being it impossible to detect as we cut the lower rows to reduce the error of the segmentation.

The second issue is the appearance of crossroads the classifier is not trained to detect. The classifier has been trained to detect crossroads where the car can go either left or right, however, if the options are to keep going straight or turning right (for example), the classifier does not have a category for it. This can be seen in figure 25.

The last limitation of the set is the difficulty to detect the road in images with strong sun, which difficulties the visibility as it can be seen in figure 26.

Even if it does not appear in the set there is also an important limitation of the system that must be taken into account. As the segmentation CNN is only trained to detect the road, it does not detect the cars in the images, considering those parts instances of the images where there is not road. This directly implies that the road classifier could not work in situations with high traffic or could show problems if a car covers a big part of the road appearing in the image.

3. ECONOMIC ASPECTS

In tables 3 and 4 it can be seen the budget of the project divided in internal hours and amortizations. As it is an investigation project there are not materials or processes that have to be taken into account. In the amortization, it has been considered not only the cost of the Matlab software itself, but also the extra toolboxes that are needed to do this particular project.

Internal hours			
Concept	**Hours**	**Unitary price (€/h)**	**Subtotal**
Control engineer	450	50	22.500 €
TOTAL			**22.500 €**

Table 3: Internal hours

Amortizations				
Concept	**Price (€)**	**Use (hours)**	**Life (hours)**	**Subtotal (€)**
MatLab license	800	275	2400	91,67 €
Computer vision toolbox license	500	250	2400	52,08 €
Image processing toolbox license	400	250	2400	41,67 €
Deep learning toolbox license	460	175	2400	33,54 €
Computer	1000	450	5000	90,00 €
TOTAL				**308,96 €**

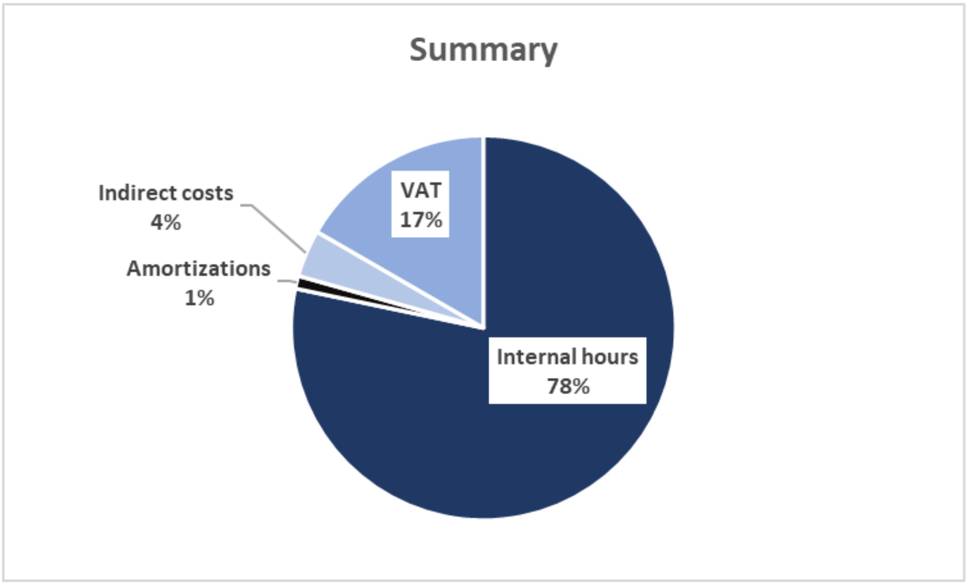
Table 4: Amortizations

By taking into account the total costs obtained before we can build table 5, which leads to the final cost of the project.

Summary	
Concept	**Cost (€)**
Internal hours	22.500,00 €
Amortizations	308,96 €
Subtotal	**22.808,96 €**
Indirect costs (5%)	1.140,45 €
VAT (21%)	4.789,88 €
TOTAL	**28.739,29 €**

Table 5: Budget summary

As it can clearly be seen, most of the costs of the project are due to the human resources. In graph 9 it can be seen what fraction of the total cost represents each of the parts.



Graph 9: Budget summary

4. CONCLUSIONS

In this project, a method for determining the car location based on camera images and aided by the IMU and the different possibilities to extract features from an image have been explained. Among them, a simple road segmentation CNN has been developed together with a classifier model to obtain the road type the car is at, with the objective of using it for relocating the vehicle in a map.

The dependence of the IMU could be corrected by developing a fixed object detection network and by applying triangulation to the detected objects, which would allow to obtain the distance from the vehicle to the object and from frame to frame, its displacement.

Also, the classifier has been trained with very few images due to the reduced time to develop the project and only four categories are introduced. An increase in the number of training images would make possible to develop a classification CNN, being this a much more robust option than the developed KNN or SVM models. The increase in the categories is necessary to recognize environments such as a crossroad where the car can go either straight or to the right.

It is clear that many of the resources of computer vision presented in the state-of-the-art (section 1.5) are very useful tools for car location systems and in general, for autonomous driving. In this project, only a small approach to those methods has been done, being the first step to obtain a robust localization system based not only in camera measurements but also in the other described sensors.

5. BIBLIOGRAPHY

[1] Erik Coelingh, Lotta Jakobsson, Henrik Lind, Magdalena Lindman. Collision warning with auto brake, a real-life safety perspective. Volvo Car corporation, Sweden, 2007.

[2] Andreas Geiger, Phillip Lenz, Christoph Stiller, Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. International Journal of Robotics Research (IJRR), 2013.

[3] "Automotive Radar Dataset for Deep Learning Based 3D Object Detection"; *Meyer, Michael and Kuschk, Georg*; European Radar Conference 2019.

[4] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, Trevor Darrell. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. UC Berkeley, Cornell University, UC San Diego, Element, Inc.

[5] https://self-driving.lyft.com/level5/data/?source=post_page---------------------------

[6] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, “nuScenes: A multimodal dataset for autonomous driving”, In arXiv preprint arXiv:1903.11027.

[7] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner, “The Oxford Radar RobotCar Dataset: A Radar Extension to the Oxford RobotCar Dataset,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Paris, 2020.

[8] https://scale.com/open-datasets/pandaset

[9] R. Hartley, and A. Zisserman, “Multiple View Geometry in Computer Vision”. Cambridge University Press, New York, NY, USA, second edition, (2003).

[10] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving. Department of Electronic Engineering, Tsinghua University. Baidu Inc.

[11] Çağrı Kaymak, Ayşegül Uçar. A Brief Survey and an Application of Semantic Image Segmentation for Autonomous Driving. Firat University, Mechatronics Eng. Dept. 23119, Elazig, Turkey.

[12] https://blogs.mathworks.com/deep-learning/2017/11/17/deep-learning-for-automated-driving-part-2-lane-detection/

[13] https://towardsdatascience.com/finding-lane-lines-simple-pipeline-for-lane-detection-d02b62e7572b

[14] Abdelmoghit Zaarane, Ibtissam Slimani, Wahban Al Okaishi, Issam Atouf, Abdellatif Hamdoun. Distance measurement system for autonomous vehicles using stereo camera. LTI Lab, Department of Physics, Faculty of Sciences Ben M'Sik, University Hassan II Of Casablanca, Morocco.

[15] Tong Qin, Peiliang Li, and Shaojie Shen. Relocalization, Global Optimization and Map Merging for Monocular Visual-Inertial SLAM.

[16] Shujun Ma, Xinhui Bai, Yinglei Wang and Rui Fang. Robust Stereo Visual-Inertial Odometry Using Nonlinear Optimization. School of Mechanical Engineering and Automation, Northeastern University, China.

[17] https://semcon.com/offerings/applied-autonomy/lidar-vs-radar-for-applied-autonomy/

[18] https://en.wikipedia.org/wiki/Support_vector_machine#Definition

[19] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#Algorithm

[20] https://www.mathworks.com/help/vision/examples/semantic-segmentation-using-deep-learning.html

[21] https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d

Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO
INGENIARITZA ESKOLA

ESCUELA
DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

METHODOLOGY FOR CAR LOCATION AND MOTION DETERMINATION USING CAMERA SENSORS AND IMAGE SEGMENTATION AND CLASSIFICATION

ANEX I – *POSITION ESTIMATOR CODE*

Estudiante	*Ander Ramos Moreno*
Director	*Mikel Díez Sánchez*
Co-Director	*Asier Zubizarreta Picó*
Departamento	*Automática y control*
Curso académico	*2019-2020*

Bilbao, 12, Septiembre, 2020

INDEX

6. ANEX I: POSITION ESTIMATOR CODE

6.1. MAIN

```
clc
close all
clear all

%Introduce image path here:
imagePath = 'J:\Script\ImageSets\0042\2011_10_03_drive_0042_extract\';

%The data corresponding to the IMU measurements is loaded
load IMU_0042.mat

%To fit the IMU measurements to the camera measurements we have to
%make the average value of each 10 measurements more or less
vx = IMU(1:end-15,1);
vy = IMU(1:end-15,2);
vz = IMU(1:end-15,3);

n=0;
k=0;
for i=1:11760
    n=n+1;
    if n>28
        n=0;
        k=k+1;
    end
    k=k+1;
    vxmod(i,1) = vx(k,1);
    vymod(i,1) = vy(k,1);
    vzmod(i,1) = vz(k,1);
end

c=0;
for h=1:1175
    vxavg=0;
    vyavg=0;
    vzavg=0;
    for r=1:10
        c=c+1;
        vxaux = vxmod(c,1);
        vxavg = vxavg+vxaux;
        vyaux = vymod(c,1);
        vyavg = vyavg+vyaux;
        vzaux = vzmod(c,1);
        vzavg = vzavg+vzaux;
    end
    vxavg = vxavg/10;
    vyavg = vyavg/10;
    vzavg = vzavg/10;
    v(h,1) = (vxavg^2+vyavg^2+vzavg^2)^0.5; %Extraction of the modulus
end

d = v*0.1; %The magnitude of the advanced distance is calculated

%The information of each camera is extracted from the dataset
set0 = strcat(imagePath,'image_00\data');
set0 = set0(find(~isspace(set0)));
set0 = imageDatastore(fullfile(set0));

set1 = strcat(imagePath,'image_01\data');
set1 = set1(find(~isspace(set1)));
set1 = imageDatastore(fullfile(set1));

set2 = strcat(imagePath,'image_02\data');
set2 = set2(find(~isspace(set2)));
set2 = imageDatastore(fullfile(set2));

set3 = strcat(imagePath,'image_03\data');
set3 = set3(find(~isspace(set3)));
set3 = imageDatastore(fullfile(set3));

%Calibration parameters
%K = Intrinsic/Calibration matrix
%D = Distorsion coefficients
K0 = [9.799200e+02 0.000000e+00 6.900000e+02;
      0.000000e+00 9.741183e+02 2.486443e+02;
      0.000000e+00 0.000000e+00 1.000000e+00]';

D0 = [-3.745594e-01 2.049385e-01 1.110145e-03 1.379375e-03 -7.084798e-
02];

K1 = [9.903522e+02 0.000000e+00 7.020000e+02;
      0.000000e+00 9.855674e+02 2.607319e+02;
      0.000000e+00 0.000000e+00 1.000000e+00]';

D1 = [-3.712084e-01 1.978723e-01 -3.709831e-05 -3.440494e-04 -
6.724045e-02];

K2 = [9.601149e+02 0.000000e+00 6.947923e+02;
      0.000000e+00 9.548911e+02 2.403547e+02;
      0.000000e+00 0.000000e+00 1.000000e+00]';
D2 = [-3.685917e-01 1.928022e-01 4.069233e-04 7.247536e-04 -6.276909e-
02];

K3 = [9.049931e+02 0.000000e+00 6.957698e+02;
      0.000000e+00 9.004945e+02 2.389820e+02;
      0.000000e+00 0.000000e+00 1.000000e+00]';

D3 = [-3.735725e-01 2.066816e-01 -6.133284e-04 -1.193269e-04 -
7.600861e-02];
```

```
%Initialization of the calculation
for j=1:4
    if j==1
        set = set0;
        K = K0;
        D = D0;
    elseif j==2
        set = set1;
        K = K1;
        D = D1;
    elseif j==3
        set = set2;
        K = K2;
        D = D2;
    else
        set = set3;
        K = K3;
        D = D3;
    end
    for p=750:1175
        Pict1 = readimage(set,p);
        Pict2 = readimage(set,p+1);
        try %If an error occurs, it displays it and continues with the
        %following frame
        [angle(p,j),loc(p,1:3,j)] =
positionEstimator(Pict1,Pict2,K,D);
        %The yaw and direction are calculated
        catch ME
        fprintf('error in one frame: %s\n', ME.message);
        end
    end
end

%Location calculation
t=loc.*d;

%Error filter
for j=1:4
    for z=750:(1175-4) %First and last part are not analyzed
        point1 = angle(z,j);
        point2 = angle(z+1,j);
        point3 = angle(z+2,j);
        if point2==0
            if point3==0
                point4 = angle(z+3,j);
                if point4==0
                    point5 = angle(z+4,j);
                    if point5==0
                        disp('Program could not detect yaw in 5
consecutive points');
                    end
                else
                    angle(z+1,j)=(point1+point4)/2;
                    angle(z+2,j)=(angle(z+1,j)+point4)/2;
                end
            else
                angle(z+1,j)=(point1+point3)/2;
            end
        else
            diff = abs(point2-point1);
            if diff>0.33 %This number represents the turn made by the
driver with the steering wheel
                point2 = (point1+point3)/2; %Here, we are assuming
that the 3rd point is correct. If we wanted
                %to make the system more robust we would have to
analyze
                %the difference between 1st and 3rd points.
                angle(z+1,j) = point2;
            end
        end
    end
end



%Data introduction in excel
T = table(angle(:,1),t(:,:,1));
filename = 'results.xlsx';
writetable(T,filename,'Sheet',4,'Range','A3','WriteVariableNames',0);

T = table(angle(:,2),t(:,:,2));
filename = 'results.xlsx';
writetable(T,filename,'Sheet',4,'Range','E3','WriteVariableNames',0);

T = table(angle(:,3),t(:,:,3));
filename = 'results.xlsx';
writetable(T,filename,'Sheet',4,'Range','I3','WriteVariableNames',0);

T = table(angle(:,4),t(:,:,4));
filename = 'results.xlsx';
writetable(T,filename,'Sheet',4,'Range','M3','WriteVariableNames',0);
```

6.2. POSITION ESTIMATOR

```
function [yaw,loc]=positionEstimator(Pict1,Pict2,K,D)

%Obtains the rotation matrix and the translation unitary vector
%between two consecutive frames

format shortg

Pict1 = editImage(Pict1); %Improve contrast and change to grey
Pict2 = editImage(Pict2);

%Create a calibration parameters object
cameraParams =
cameraParameters('intrinsicMatrix',K,'radialDistortion',...
    [D(1) D(2) D(5)],'tangentialDistortion',[D(3) D(4)]);

%Undistort images
Pict1 = undistortImage(Pict1, cameraParams);
Pict2 = undistortImage(Pict2, cameraParams);

%Extract features by SURF algorithm
Corners1 = detectSURFFeatures(Pict1,'MetricThreshold', 500);
numPoints = 1000;
Points1 = selectUniform(Corners1, numPoints, size(Pict1));

% Extract features. Using 'Upright' features improves matching quality
if
% the camera motion involves little or no in-plane rotation.
Features1 = extractFeatures(Pict1, Points1,'Upright', true);

%Extract and match with second image
[Points2, Features2, indexPairs] = helperDetectAndMatchFeatures...
    (Features1, Pict2);

% Estimate the pose of the current view relative to the previous view.
[orient, loc, inlierIdx] = estimateRelativePosition_Essential(...
    Points1(indexPairs(:,1)), Points2(indexPairs(:,2)), cameraParams);

angles = rotm2eul(orient); %Extract angles from rotation matrix
yaw = angles(2)*180/pi; %Change from radians to degrees

end
```

6.3.EDIT IMAGE

```
function [Pict]=editImage(Pict)

%For improving contrast
Pict = histeq(Pict);

%If color image, changes to grey
if size(Pict,3)==3
    Pict = rgb2gray(Pict);
end

end
```

6.4. Estimate relative position

```
function [orientation, location, inlierIdx] = ...
    estimateRelativePosition(matchedPoints1, matchedPoints2,
cameraParams)

if ~isnumeric(matchedPoints1)
    matchedPoints1 = matchedPoints1.Location;
end

if ~isnumeric(matchedPoints2)
    matchedPoints2 = matchedPoints2.Location;
end

%We try a hundred times to estimate the essential matrix
for i = 1:100
    % Estimate the essential matrix.
    [E, inlierIdx] = estimateEssentialMatrix(matchedPoints1,
matchedPoints2,...
        cameraParams,'MaxNumTrials',1000);

    % Make sure we get enough inliers, if not, recalculate
    if sum(inlierIdx) / numel(inlierIdx) < .3
        continue;
    end

    % Get the epipolar inliers.
    inlierPoints1 = matchedPoints1(inlierIdx, :);
    inlierPoints2 = matchedPoints2(inlierIdx, :);

    % Compute the camera pose from the fundamental matrix
    [orientation, location, validPointFraction] = ...
        relativeCameraPose(E, cameraParams, inlierPoints1(1:2:end,
:),...
        inlierPoints2(1:2:end, :));

    % validPointFraction is the fraction of inlier points that project
in
    % front of both cameras. If the this fraction is too small, then
the
    % fundamental matrix is likely to be incorrect (recalculate).
    if validPointFraction > .8
       return;
    end
end

% After 100 attempts validPointFraction is still too low.
error('Unable to compute the Essential matrix');
```

BILBOKO
INGENIARITZA ESKOLA

ESCUELA
DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

METHODOLOGY FOR CAR LOCATION AND MOTION DETERMINATION USING CAMERA SENSORS AND IMAGE SEGMENTATION AND CLASSIFICATION

ANEX II – *ROAD CLASSIFIER CODE*

Estudiante	*Ander Ramos Moreno*
Director	*Mikel Díez Sánchez*
Co-Director	*Asier Zubizarreta Picó*
Departamento	*Automática y control*
Curso académico	*2019-2020*

Bilbao, 12, Septiembre, 2020

INDEX

7. ANEX II: ROAD CLASSIFIER CODE

7.1. COUNTER ROAD DELETE

```
clc
close all
clear all

%Changes the labeled pixels as "counter lane" (black) to
%the "rest" category (red)

%Introduce image folder path
labDir=fullfile('G:\TFM\Script\6-Road\Kitti\data_road\tr\lab');
tsds=imageDatastore(labDir);

labDirAux = dir('G:\TFM\Script\6-Road\Kitti\data_road\tr\lab\*.png');

Red = [255 0 0];
Black = [0 0 0];

for z=1:289 %Set here the number of labels
    im = readimage(tsds,z);
    for i=1:375
        for j=1:1242
            if im(i,j,:)==Black
                im(i,j,:) = Red;
            end
        end
    end

    filename = strcat('G:\TFM\Script\6-
Road\Kitti\data_road\tr\lab_2cat\'...
    ,labDirAux(z).name);
    imwrite(im,filename,'png');
end
```

7.2. SEGMENTATION CNN TRAINING

```
clc
close all
clear all

%images
imgDir = fullfile('G:\TFM\Script\6-Road\Kitti\data_road\tr\img');
imds = imageDatastore(imgDir);

classes = [
    "Road"
    "Rest"];

%labels
labelDir = fullfile('G:\TFM\Script\6-Road\Kitti\data_road\tr\lab');
Red = [255 0 0;];
Pink = [255 0 255;];
pxds = pixelLabelDatastore(labelDir,classes,{Pink Red});

 % Specify the network image size. This is typically the same as the
training image sizes.
 imageSize = [375 1242 3];

% Specify the number of classes.
numClasses = numel(classes);

net = resnet18;

%Create net
lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");

%Set augmenter to increase training info
pixelRange = [-30 30];
augmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);

%Create image and label datastore
pximds = pixelLabelImageDatastore(imds,pxds, ...
    'DataAugmentation',augmenter);

% Define training options.
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2,...
    'Plots','training-progress');


[net, info] = trainNetwork(pximds,lgraph,options);
```

7.3. PARAMETER EXTRACTION (FOR TRAINING AND FOR OBTAINING RESULTS)

```
clc
close all
clear all

load 'RoadDetector2.mat' %Introduce segmentation net name
load 'Model5.mat' %Introduce predictor name

%Introduce image path
testDir=fullfile('J:\TFM\Calculos\Script\ImageSets\0009\2011_09_26_dri
ve_0009_extract\image_02\data'); %Introduce images folder path
tsds=imageDatastore(testDir);

Initializer = 0;
    for s=1:453 %Introduce the number of pictures in file

        disp(['Picture ',num2str(Initializer+s)])

        I = readimage(tsds,Initializer+s); %Read image

        %Adjustment to image size
        I = imresize(I,[375,1242]);
        [height,wide,channels] = size(I);

        C = semanticseg(I, net); %Segment image

        B = labeloverlay(I,C,'Transparency',0.4); %Visualize segmented
image
% %         figure
% %         imshow(B)



        %Post-process

        Iaux = rgb2gray(I); %Create auxiliar black and white image

        %Cut the lower part of the image where distortion is high
        height=height-60;
        Iaux = imcrop(Iaux,[0 0 wide height]);

        for i=1:height %Paint road white and rest in black
            for j=1:wide
                if C(i,j)=='Road'
                    Iaux(i,j) = 1000;
                else
                    Iaux(i,j) = 0;
                end
            end
        end

        Iaux = imgaussfilt(Iaux); %Apply Gauss filter to smooth

        Iaux = bwareaopen(Iaux,15000); %Delete pixel groups
        %smaller than 15000, changes to logical

        %pixelAmount calculation
        pixelAmount=0;
        for i=1:height
            for j=1:wide
                if Iaux(i,j)==1
                    pixelAmount = pixelAmount+1;
                end
            end
        end

        %Calculation of highest point in image of road
        i=1;
        j=1;
        while Iaux(i,j)==0
            if j==wide
                j=0;
                i=i+1;
            end
            if i==height
                maxHeight = i;
                break
            end
            maxHeight = i;
            j=j+1;
        end

        %If there is not detection, we introduce a zero in all
parameters
        if maxHeight == height
            Var1(s,1) = maxHeight;
            Var2(s,1) = 0;
            Var3(s,1) = 0;
            Var4(s,1) = 0;
            Var5(s,1) = 0;
            Var6(s,1) = 0;
            Var7(s,1) = 0;
            Var8(s,1) = 0;
            Var9(s,1) = 0;
            Var10(s,1) = 0;
            Var11(s,1) = 0;
            Var12(s,1) = 0;
            Var13(s,1) = 0;
            Var14(s,1) = 0;
            Var15(s,1) = 0;
            Var16(s,1) = 0;
            Var17(s,1) = 0;
            Var18(s,1) = 0;
            Var19(s,1) = 0;
            continue
        end

        %Region of analysis definition
        regionTop = height-round((height-maxHeight)*0.8);
        regionBottom = height;

        %Calculation of middle point in lower row
        r=0;
        avg=0;
        counter=0;
        for j=1:wide
            if Iaux(height,j) == 1
                avg=avg+j;
                counter=counter+1;
            end
        end
        if avg ~= 0
            avg=round(avg/counter);
        else
            disp('ERROR: avg=0');
        end


        %Left and right points array obtention

        %Left
        z=0;
        for i=regionTop:regionBottom
            z=z+1;
            for j=1:wide
                if Iaux(i,j)==1
                    leftPoints(z,1)=j;
                    leftPoints(z,2)=-i;
                    break
                end
            end
            leftDistance(z,1)=avg-j;
        end

        %Right
        z=0;
        for i=regionTop:regionBottom
            z=z+1;
            for j=1:wide
                if Iaux(i,wide+1-j)==1
                    rightPoints(z,1)=wide+1-j;
                    rightPoints(z,2)=-i;
                    break
                end
            end
            rightDistance(z,1)=wide+1-j-avg;
        end

        batch = round(z/4);

        %Central points array
        avgPoints(:,2) = -(regionTop:regionBottom);
        avgPoints(:,1) = avg;


        %Line approximation and angle calculation of left and right
points
        leftLine = fit(leftPoints(:,1),leftPoints(:,2),'poly1');
        rightLine = fit(rightPoints(:,1),rightPoints(:,2),'poly1');
        leftAngle = angleCalculation(leftLine);
        rightAngle = angleCalculation(rightLine);

%         %Point representation
%         figure
%         scatter(leftPoints(:,1),leftPoints(:,2));
% %         hold on
% %         plot(leftLine)
%         hold on
%         scatter(rightPoints(:,1),rightPoints(:,2));
% %         hold on
% %         plot(rightLine)
%         hold on
%         scatter(avgPoints(:,1),avgPoints(:,2));


%         %Border representation(not used for calculations)
%         Iaux = edge(Iaux,'Canny',0.2,0.5); %Extract white and black
edges
%         Iaux=bwareaopen(Iaux,10);
%         Iaux=imdilate(Iaux,true(3));
%
%         figure
%         imshow(Iaux)


        %Average values of left and right distances
        leftAvgDistance = mean(leftDistance);
        leftDistance1 = leftDistance(1,1);
        leftDistance2 = leftDistance(1+batch,1);
        leftDistance3 = leftDistance(1+2*batch,1);
        leftDistance4 = leftDistance(1+3*batch,1);
        leftDistance5 = leftDistance(end,1);
        rightAvgDistance = mean(rightDistance);
        rightDistance1 = rightDistance(1,1);
        rightDistance2 = rightDistance(1+batch,1);
        rightDistance3 = rightDistance(1+2*batch,1);
        rightDistance4 = rightDistance(1+3*batch,1);
        rightDistance5 = rightDistance(end,1);
        %Parameter introduction
        Var1(s,1) = maxHeight;
        Var2(s,1) = avg;
        Var3(s,1) = leftAngle;
        Var4(s,1) = rightAngle;
        Var5(s,1) = leftAvgDistance;
        Var6(s,1) = leftDistance1;
        Var7(s,1) = leftDistance2;
        Var8(s,1) = leftDistance3;
        Var9(s,1) = leftDistance4;
        Var10(s,1) = leftDistance5;
        Var11(s,1) = rightAvgDistance;
        Var12(s,1) = rightDistance1;
        Var13(s,1) = rightDistance2;
        Var14(s,1) = rightDistance3;
        Var15(s,1) = rightDistance4;
        Var16(s,1) = rightDistance5;
        Var17(s,1) = pixelAmount;
        relationHeightDistanceLeft = leftAvgDistance/maxHeight;
        Var18(s,1) = relationHeightDistanceLeft;
        relationHeightDistanceRight = rightAvgDistance/maxHeight;
        Var19(s,1) = relationHeightDistanceRight;

        %Introducing parameters in table and applying sliding window.
        %This is done when calculating, not when extracting
information
        %for training models
        if s>4
            Taux = table(Var1(end-4:end,1),Var2(end-4:end,1),Var3(end-
4:end,1)...
                ,Var4(end-4:end,1),Var5(end-4:end,1),Var6(end-
4:end,1),Var7(end-4:end,1)...
                ,Var8(end-4:end,1),Var9(end-4:end,1),Var10(end-
4:end,1),Var11(end-4:end,1)...
                ,Var12(end-4:end,1),Var13(end-4:end,1),Var14(end-
4:end,1),Var15(end-4:end,1)...
                ,Var16(end-4:end,1),Var17(end-4:end,1),Var18(end-
4:end,1),Var19(end-4:end,1));
            Taux.Properties.VariableNames =
{'maxHeight','avg','leftAngle',...

'rightAngle','leftAvgDistance','leftDistance1','leftDistance2',...

'leftDistance3','leftDistance4','leftDistance5','rightAvgDistance',...

'rightDIstance1','rightDistance2','rightDistance3','rightDistance4',..
.

'rightDistance5','pixelAmount','relationLeft','relationRight'};
            result = Model5.predictFcn(Taux);

            finalResult(s,1) = pastFrames(result)
        else
```

```
            Taux = table(Var1,Var2,Var3...
                ,Var4,Var5,Var6,Var7...
                ,Var8,Var9,Var10,Var11...
                ,Var12,Var13,Var14,Var15...
                ,Var16,Var17,Var18,Var19);
            Taux.Properties.VariableNames =
{'maxHeight','avg','leftAngle',...

'rightAngle','leftAvgDistance','leftDistance1','leftDistance2',...

'leftDistance3','leftDistance4','leftDistance5','rightAvgDistance',...

'rightDIstance1','rightDistance2','rightDistance3','rightDistance4',..
.

'rightDistance5','pixelAmount','relationLeft','relationRight'};
            result = Model5.predictFcn(Taux);
            if result=='Straight'
                finalResult(s,1)='S';
            elseif result=='Crossroad'
                finalResult(s,1)='C';
            elseif result=='Left curve'
                finalResult(s,1)='L';
            else
                finalResult(s,1)='R';
            end
        end


        clear avgPoints rightPoints leftPoints maxheight avg leftAngle
rightAngle...
            leftDistance rightDistance leftDistance1 rightDistance1
leftDistance2 rightDistance2...
            leftDistance3 rightDistance3 leftDistance4 rightDistance4
leftDistance5 rightDistance5...
            batch Taux result
    end


% %Introduction of database in Excel for training
% T =
table(Var1,Var2,Var3,Var4,Var5,Var6,Var7,Var8,Var9,Var10,Var11,Var12,V
ar13,Var14,Var15,Var16,Var17,Var18,Var19);
% filename = 'imageInfo.xlsx';
% initialCell = 'F88';
%
writetable(T,filename,'Sheet',1,'Range',initialCell,'WriteVariableName
s',0);


%Introduction of data in Excel for results
Taux = table(Var1,Var2,Var3...
    ,Var4,Var5,Var6,Var7...
    ,Var8,Var9,Var10,Var11...
    ,Var12,Var13,Var14,Var15...
    ,Var16,Var17,Var18,Var19);

T = table(finalResult);
filename = 'imageInfo.xlsx';
initialCell = 'D2';
sheetNumber = 23;
writetable(T,filename,'Sheet',sheetNumber,'Range',initialCell,'WriteVa
riableNames',0);
initialCell = 'E2';
writetable(Taux,filename,'Sheet',sheetNumber,'Range',initialCell,'Writ
eVariableNames',0);
```

7.4. FLIPER

```
clc
close all
clear all

%Flips images to duplicate the training information

%Introduce image path
trainDir=fullfile('J:\TFM\Calculos\Script\6-
Road\Kitti\data_road\imageInfoDatabase\training images');
trds=imageDatastore(trainDir);

trDir=dir('J:\TFM\Calculos\Script\6-
Road\Kitti\data_road\imageInfoDatabase\training images\*.png');


    for s=1:88 %Introduce the number of pictures

        disp(['Picture ',num2str(s),':'])

        I = readimage(trds,s); %Read image

        I = flipdim(I ,2);

        A = strcat('fliped',trDir(s).name);

        newname = A(find(~isspace(A)));

        newfilename=strcat('J:\TFM\Calculos\Script\6-
Road\Kitti\data_road\imageInfoDatabase\training images flipped\'...
            ,newname);
        imwrite(I,newfilename,'png');
    end
```