Universidade de Aveiro Departamento de Engenharia Mecânica
2014

Rui Filipe Cabral de Azevedo

Sensor Fusion of LASER and Vision in Active Pedestrian Detection

Deteção Ativa de Peões por Fusão Sensorial de LASER e Visão

Rui Filipe Cabral de Azevedo

Sensor Fusion of LASER and Vision in Active Pedestrian Detection

Deteção Ativa de Peões por Fusão Sensorial de LASER e Visão

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica de Vítor Manuel Ferreira dos Santos, Professor do Departamento de Engenharia Mecânica da Universidade de Aveiro

O júri / The jury

Presidente / President	**Professor Doutor Jorge Augusto Fernandes Ferreira** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Professor Doutor Paulo Miguel de Jesus Dias** Professor Auxiliar do Universidade de Aveiro - DETI
	Professor Doutor Vítor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que me ajudaram, direta ou indiretamente, durante este 5 anos e colaboraram para o desfecho desta formação académica.

Ao meu Orientador, Professor Doutor Vítor Manuel Ferreira dos Santos, pela confiança, criticas e apoio que sempre me deu ao longo deste projecto, estando assim apto para agarrar a vida pelos cornos.

Aos meus pais, Alcindo e Fatinha, a oportunidade que me deram para prosseguir os meus estudos e apoio incondicional em todos os anos da minha vida, cujo amor incondicional sempre foi e será um autêntico pilar na minha vida. MUITO OBRIGADO por tudo.

Aos meus padrinhos e prima, que sempre acreditaram que eu seria capaz e sempre me acolheram com um sorriso.

Às minhas grandes avós, que me ensinaram o que é trabalhar na terra e mostraram-me o quanto a vida pode ser difícil, mas não impossível.

À minha melhor amiga, Luísa, por toda a paciência, carinho, atenção e apoio que sempre me deu. Obrigado por me ajudares a ultrapassar todos os obstáculos.

Aos meus verdadeiros amigos, aos amigos de mecânica e aos meus colegas do LAR, o meu muito obrigado pelos bons momentos que passamos e de certeza que vamos continuar a passar :). Queria também agradecer especialmente ao grande Jorge Almeida, que nunca deixou de parar o que quer que fosse para me ajudar.

A todos um Muito Obrigado!

Palavras-chave Deteção de peões, Fusão sensorial, LIDAR, Classificação de possiveis peões, Multi-thread, Calibração

Resumo Este trabalho explora uma técnica de fusão sensorial que visa dotar veículos de mecanismos rápidos de detecção de peões em ambiente exterior. O método restringe as zonas de procura numa imagem com base em indicadores obtidos por outro sensor (LIDAR). Esta técnica tem como base a idéia de que havendo um registo entre os sensores envolvidos, um sensor "rápido" mas pouco preciso, pode indicar as regiões onde potencialmente há alvos, e outro sensor, "lento" mas mais robusto, é utilizado para fazer a confirmação da deteção. Com vista a explorar essas propriedades, foi criado um algoritmo que utiliza a informação de dois sensores, para primeiro selecionar, de entre muitos objectos, possíveis peões(fase LIDAR) e dada a informação da localização do possível pedestre, uma imagem já à escala e precisa da localização, é recortada da imagem inicial, sendo a mesma enviada a ser processada por um detetor de peões (sensor mais robusto), permitindo a sua rigorosa classificação. O método é testado em dois conjuntos de dados diferentes e os resultados confirmam a sua validade.

Key words Pedestrian Detection, Sensor Fusion, LIDAR, Possible Pedestrian Classification, Multi-thread, Calibration

Abstract This work explores a technique of sensor fusion that aims to equip vehicles with pedestrian fast detection mechanisms in exterior environments. This method restricts image areas of search based on indicators obtained by another sensor (LIDAR). This technique is based on the idea that when having a registration among the involved sensors, one "fast" sensor, but inaccurate, that can indicate regions where potential pedestrian are located on the image, and another sensor, "slower" but more robust that is used to confirm detection more accurately. So, an algorithm was created to merge two algorithms, a LIDAR-based tracking and a vision-based detection algorithm; The LIDAR indicates the precise location and scale of the potential pedestrian on the image, and crop the image relative to the potential pedestrian, being processed afterwards by one pedestrian detection algorithm to validate the classification. The method is tested in two different cases and the results confirm their validity.

Contents

Contents i

List of Figures iii

List of Tables v

List of Acronyms vii

1 Introduction 1
- 1.1 ATLAS Project . . . 1
- 1.2 Motivation . . . 1
- 1.3 Objectives . . . 2
- 1.4 State of the Art . . . 2
  - 1.4.1 LIDAR-based systems . . . 2
  - 1.4.2 Vision-based pedestrian detection . . . 5
  - 1.4.3 Sensor Fusion Information Systems . . . 7
  - 1.4.4 Calibration Methods . . . 9
  - 1.4.5 Discussion . . . 10

2 Experimental Setup 11
- 2.1 Software and Hardware Tools . . . 11
  - 2.1.1 Software . . . 11
  - 2.1.2 Hardware . . . 12
- 2.2 Calibration Methods . . . 13
  - 2.2.1 Manual calibration tool . . . 13
  - 2.2.2 Automatic calibration tool . . . 17
  - 2.2.3 Discussion . . . 19

3 Algorithm for Active Pedestrian Detection 21
- 3.1 Re-Parametrization of Existing Pedestrian Detection Algorithm . . . 22
- 3.2 Sensor fusion algorithm . . . 22
  - 3.2.1 LIDAR Classification of Potential Pedestrian . . . 22
  - 3.2.2 Active Pedestrian Detection . . . 24

4 Experiments and Results 29
- 4.1 Experiment 1 - Tripod mounting indoors . . . 31
- 4.2 Experiment 2 - Tripod mounting outdoors . . . 33

4.3 Experiment 3 - AtlasCar 38
4.4 Discussion 43

5 Conclusions and Future Work **45**

References **47**

A Usage Instructions **49**
A.1 Sensor Fusion Program 49
A.2 Calibration Process 51

B Hardware and System Notes **53**

List of Figures

1.1 Atlas 2010 (left) and Atlas MV (right) . . . 1
1.2 AtlasCar . . . 1
1.3 The object models for pedestrians, bicycle and car[Zhao et al., 2009] . . . 3
1.4 Pedestrian detection algorithm using a multilayer laser sensor[Gidel et al., 2009] 3
1.5 Laser ranger finder . . . 4
1.6 LASER localization on the environment (left), moving target tracking (right) . 4
1.7 In this illustration, the black rectangles represent the DW over the 10 channels, and the red rectangles represent examples of random rectangles over which local sums are calculated. . . . 5
1.8 Different approaches to detecting pedestrians at multiple scales (Number of scales - N, Reducing factor - K). [Benenson et al., 2012] . . . 6
1.9 Weak classifiers aggregation . . . 7
1.10 Shadow area locations[Broggi et al., 2009]. . . . 8
1.11 Pedestrian partially hidden by parked cars [Broggi et al., 2009]. . . . 9
1.12 Calibration Tool Example: Rectangular frame with one diagonal connection [Li et al., nd] . . . 9
1.13 Feature points extraction and calibration processing [Guan et al., 2009]. . . . 10

2.1 Simple ROS Scheme communication architecture . . . 12
2.2 Hardware Cameras used . . . 13
2.3 Sick LMS 151 [Sick] . . . 13
2.4 Sick LMS 151 and Logitech HD c310 webcam tripod assembly . . . 14
2.5 Manual Calibration Example: 3,8m / 6m / 10m from the LIDAR . . . 14
2.6 SolvePnP, OpenCV function (Inputs above / outputs below) . . . 15
2.7 XB3 and Front Bumper laser localization in the car . . . 16
2.8 AtlasCar Manual Calibration Example . . . 16
2.9 Camera view, laser view and XB3 location frame in world frame . . . 17
2.10 Object used to automatize the calibration process . . . 18
2.11 Automatic Calibration Result Image . . . 19

3.1 APD resume . . . 21
3.2 Pedestrian Detection Algorithm Scheme (before on left/ after on the right) . . . 23
3.3 Potential Pedestrian Search Example shown in RViz, with the correspondent image on the lower left . . . 23
3.4 Sensor Fusion LIDAR Process Scheme . . . 24
3.5 Sensor Fusion LIDAR-Camera Scheme . . . 25

3.6 APD candidate classification process 26
3.7 APD candidate visualization process 27
3.8 Different types of Candidate classifications, Not Processed (Unknown), Processing, Pedestrian, Object (Non-Pedestrian), left to right respectively. 28

4.1 Example of output image and results of a Pedestrian classification 30
4.2 Ground truth for the same image as in figure 4.1 30
4.3 LIDAR objects shown, without any previous selection 31
4.4 Differences between candidates, Object/Pedestrian/Processing 32
4.5 Figure shows last modification made to the algorithm 33
4.6 Tripod Assembly Spot Location on the UA Campus, 22 - Mechanical Engineering Department [UA] 34
4.7 Experiment 2, TPR (True Positive Rate) and FPR (False Positive Rate) Results 34
4.8 Example Result 1 - TP and TN classifications. 36
4.9 Example Result 2 - Target tracking allows Pedestrians to maintain the same ID, even when occluded by a column. 36
4.10 Example Result 3 - Evidence shows that the tracking gets confused with some people crossing (3 pedestrians and one column -4 objects in total-). Moreover, it appears to be very complex for the further objects to be correctly tracked, since the surrounding BB of the column has now a pedestrian ahead, which induces into a classification error, switching it from "Object" to "Pedestrian", figure 4.11. The nearest pedestrians continued to be well tracked. 37
4.11 Example Result 4 - Outcome of fig. 4.10, previous Pedestrian 18 became Pedestrian 46, reclassified after receiving a new ID. 37
4.12 FP example using Pedestrian Detection Algorithm 38
4.13 AtlasCar recording location, on the UA Campus, 3 - CESAM/CICECO/TEMA Department [UA] 39
4.14 Experiment 3, TPR (True Positive Rate) and FPR (False Positive Rate) Results 39
4.15 Pedestrians Route in *AtlasCar* dataset 40
4.16 TP Fusion Algorithm Example 1 41
4.17 TP Fusion Algorithm Example 2 41
4.18 TP Fusion Algorithm Example 3 42
4.19 FP Fusion Algorithm Example - After a pedestrian passed the bushes, the classification continued to be shown for a short period of time 42
4.20 FP Pedestrian Detection Algorithm Example 43

List of Tables

2.1 Intrinsic camera parameters: K 15
2.2 Calibration results (Manual vs Automatic) 19

4.1 Terminology and definitions for performance metrics 29
4.2 Experiment 2 detailed results 35
4.3 Mean time (in seconds) to process an image 35
4.4 Experiment 3 detailed results 40
4.5 Mean time (in seconds) to process an image 43

List of Acronyms

LAR Laboratório de Automação e Robótica (Automation and Robotics Laboratory)

ADAS Advanced Driver Assistance Systems

LIDAR Light Detection And Ranging

RADAR Radio Detection And Ranging

MLE Maximum-Likelihood Estimation

ROI Region Of Interest

SVM Support Vector Machine

LDA Linear discriminant analysis

NBC Naïve Bayes Classifier

GMM Gaussian Mixture Models

ANN Artificial Neural Network

MTT Multi Target Tracking

CRF Conditional Random Field

HD High Definition

FPS Frames per second

BB Bounding Box

Chapter 1

Introduction

1.1 ATLAS Project

ATLAS is a project created and developed at the Laboratório de Automação e Robótica (Automation and Robotics Laboratory) (LAR) from the Department of Mechanical Engineering of the University of Aveiro, Portugal.

The project took its first steps in 2003 with the purpose of competing in autonomous driving competitions taking place at the Portuguese National Robotics Festival. Since then, ATLAS has been growing, with the development of several series of robots, having had a continuous success in those competitions and won many prizes; the project is divided into 2 types, two small robots (figure 1.1) and one car size robot (figure 1.2).

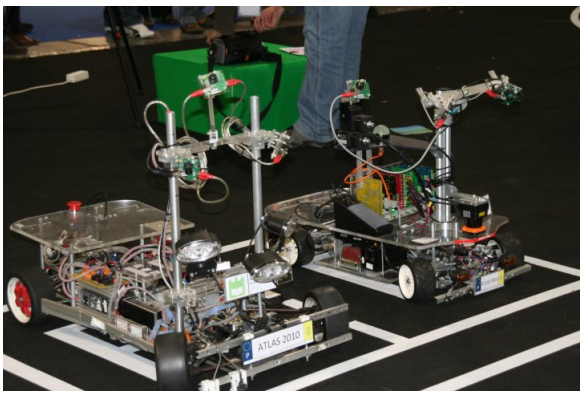


Figure 1.1: Atlas 2010 (left) and Atlas MV (right)

Figure 1.2: AtlasCar

1.2 Motivation

In modern Advanced Driver Assistance Systems (ADAS) there is a rising concern about attaching environment detection mechanisms into vehicles, particularly on the traffic agents detection, like pedestrians and other vehicles.

Pedestrians are particularly important because of security issues and the complexity of the solution of the main problem. A lot of research has been done in this area, using different

approaches, like visual and thermal image, Light Detection And Ranging (LIDAR), Radio Detection And Ranging (RADAR), etc. However, there is no definitive solution and the problem is still being studied by many researchers, from universities to automotive industries.

The use of visual image, for example, is a low cost solution for this matter, but it still has a very low efficiency, especially in image processing time, which limits real-time application.

On the other hand, there are other sensors like LIDAR that have a fast performance, but with very uncertain results on the assessment of the detected targets. Therefore, combining the information of these two different sensors, might be a way to obtain a more efficient and robust system.

The approach can be to explore one technique that narrows searching areas in an image, using indicators obtained by another sensor (LIDAR). In other words, combine the sensors involved, one "fast" sensor not very accurate, but which can indicate areas with potential targets, and another sensor, this time a "slow" one, more robust, which can be used to confirm detection.

1.3 Objectives

The main objectives of this work are:

- Adaptation of an existing visual pedestrian detection algorithm, so it can operate with different parameters.
- Development of a manual or semi-automatic calibration tool, between LIDAR and a single camera.
- Development of an application for pedestrian active detection, with image localized search, combining LIDAR and vision data.

1.4 State of the Art

This section will focus on four main points: the use of LIDAR data to easily separate objects in the environment; pedestrian classification resorting to vision image; sensor fusion classification methods and lastly, the most common automatic calibration methods.

1.4.1 LIDAR-based systems

Pedestrian detection systems based on data provided by a 2D LIDAR have different types of applications such as detection, tracking, classification and high-level decision systems. Some related work is described below.

Detection and Classification

Premebida says the LIDAR-based classification stage consists of a decision-making function which decides the class(PED or nPED) a given detected object belongs to [Premebida, 2012].

Concerning detection using only laser data, [Zhao et al., 2009] propose a ego-vehicle localization and object detection system in urban environments. In the case of cars, the data appearance varies dramatically because the relative position and direction to sensor change. They call each edge an ”axis”. In different cases where only the rear or front side of a car is

measured, a motion vector vertical to the extracted axis is detected, making up an additional axis, thus a car is characterized as a two axis object.

For bicycles, no obvious axis could be detected when the laser is measuring its rear or head, it only could fit a line when seen from the side, characterizing a bicycle as a one axis object.

In the pedestrians case, no axis can be extracted, because the appearance (resembles to semi-circle) does not change dramatically; therefore, it was characterized as a zero-axis object. The object models are shown in figure 1.3.

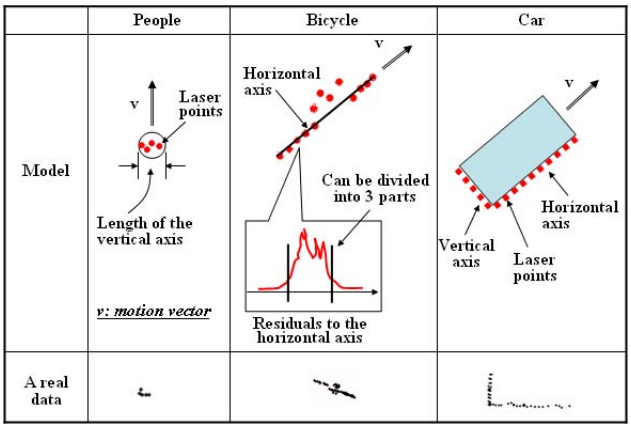


Figure 1.3: The object models for pedestrians, bicycle and car[Zhao et al., 2009]

For all laser-segments, the objects' speeds are estimated and, for pedestrians classification, a set of features is used to model the pedestrian-likelihood.

Other authors use a 4-layer LIDAR (Ibeo Alasca XT) to perform, in each laser-layer a segmentation, then each layer is fused and pedestrians are detected based on Maximum-Likelihood Estimation (MLE) which is modeled using Parzen method [Parzen, 1962]. Figure 1.4 shows a scheme used in [Gidel et al., 2009] of how the multilayer works.

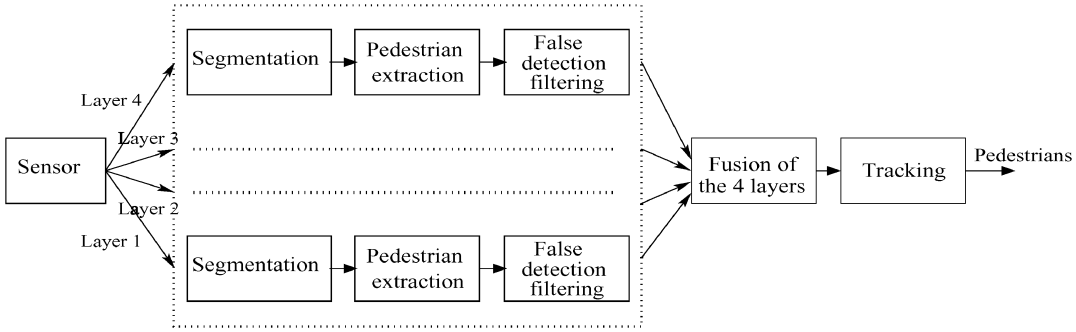


Figure 1.4: Pedestrian detection algorithm using a multilayer laser sensor[Gidel et al., 2009]

Multi Target Tracking

Much work has been developed on the subject of multi target tracking over the past few decades [Blackman and Popoli, 1999]. This is a very important matter, because normally LIDAR data are much faster than vision, therefore this technique is suited for detection and tracking of multiple moving targets in situations of strong occlusion. The process starts with the application of temporal filters to the raw data in order to remove noise, followed by a multi phase segmentation with the goal of overcoming occlusions. The resulting segments represent objects in the environment. For each segment a representative point is defined. This point is calculated to better represent the object while keeping some invariance to rotation and shape changes. In order to perform the tracking, a list of objects to follow is maintained and all visible objects are associated with objects from this list, using search techniques based on the predicted motion of objects. A search zone shaped as an ellipse is defined for each object and it is in this zone that the association is preformed. The motion prediction is based on two motion models, one with constant velocity and the other with constant acceleration and in the application of Kalman filters. The algorithm was tested in diverse real conditions and shown to be robust and effective in the tracking of people even in situations of long occlusions [Almeida, 2010].

The tests were made with the help of the laser ranger finder shown in figure 1.5; the moving targets are being tracked since they were firstly detected by the laser without losing the respective ID of the target (figure 1.6).

Figure 1.5: Laser ranger finder

Figure 1.6: LASER localization on the environment (left), moving target tracking (right)

1.4.2 Vision-based pedestrian detection

Visual detection of humans is a field with an extensive range of applications such as robotics, entertainment, surveillance, care for the elderly and disabled, road safety and others. The benefits of this become obvious when thinking about a car being driven in an urban scenario. If the circumstances are such that the driver is always aware of the surrounding pedestrians, the danger of accidents involving them would most likely decrease dramatically.

Feature extraction

Sliding window detection is a technique performed by applying a Detection Window (DW) over the image, evaluating a set of features, and then sliding it to an adjacent place to repeat the process. The DW has constant dimensions (64x128 in most sliding window detection methods), so, in order to find pedestrians with different sizes, the image has to be rescaled and reanalyzed multiple times. An image channel is a representation of the original, where the output pixels are obtained by using linear or non-linear transformations on the input ones, thus preserving the overall image layout. To make this clearer, an illustration of 20 possible random features is shown in figure 1.7.

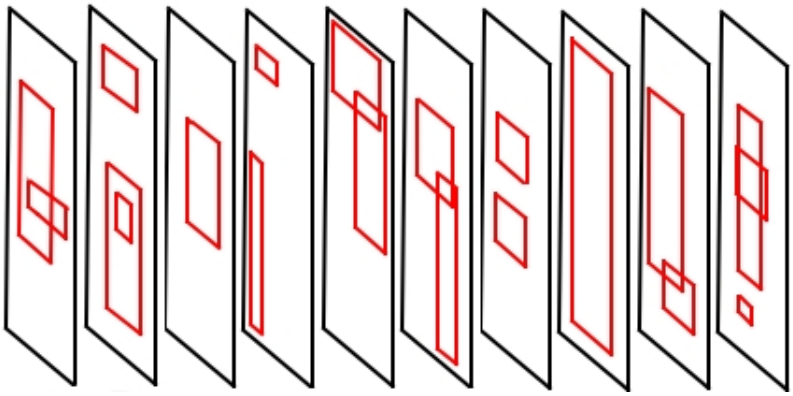
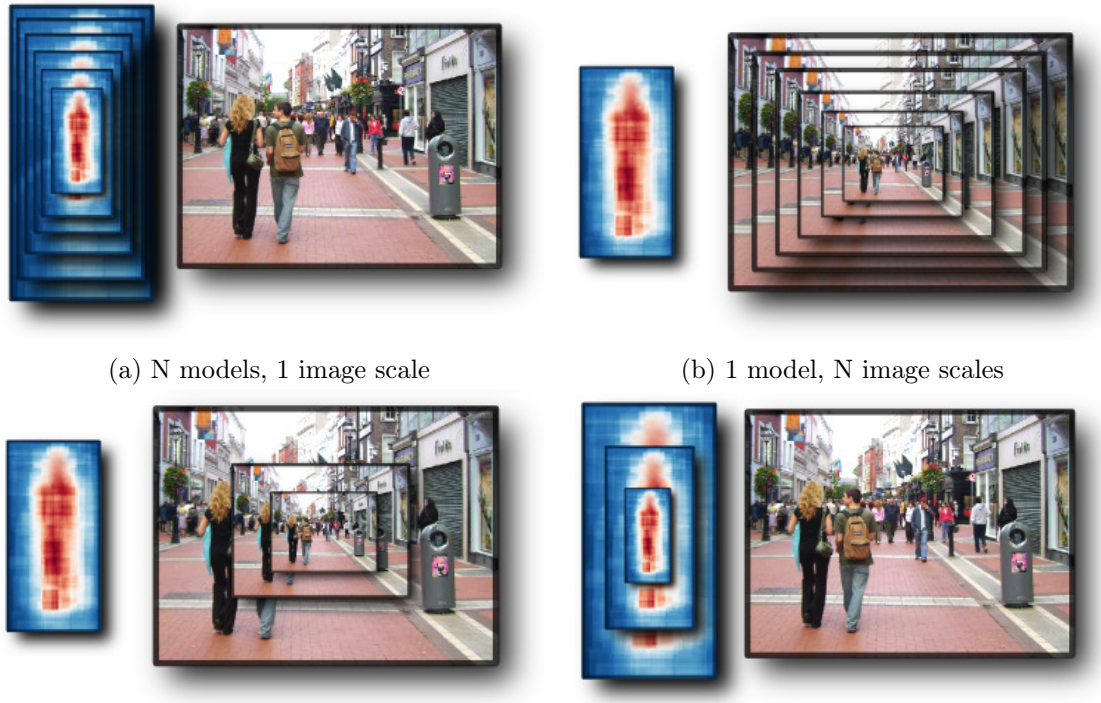


Figure 1.7: In this illustration, the black rectangles represent the DW over the 10 channels, and the red rectangles represent examples of random rectangles over which local sums are calculated.

To extract any number of random features, 5 parameters are necessary to define a rectangle: Channel index, Width, Height, X coordinate of the upper-left corner of the rectangle and Y coordinate of the upper-left corner of the rectangle;

Multi-scale Image Analysis

Once the feature extraction from DW's is set, the next step is to build an architecture for full image analysis. There are different types of approach to detect pedestrians at multiple scales, (figure 1.8).



(a) N models, 1 image scale

(b) 1 model, N image scales

(c) 1 model, $\frac{N}{K}$ image scales

(d) $\frac{N}{K}$ models, 1 image scale

Figure 1.8: Different approaches to detecting pedestrians at multiple scales (Number of scales - N, Reducing factor - K). [Benenson et al., 2012]

Explaining the images (top to bottom, left to right) each paragraph explains each image.

Assuming that object appearance is invariant to translations in the image, to implement the "N models, 1 image scale" approach, one should train as many models as there are scales, see figure 1.8a. The number of scales N is usually in the order of ∼50 scales.

The analysis is made by rescaling the image multiple times, to find pedestrians with different sizes. This process is called dense image pyramid, (fig 1.8b). A few parameters are needed to build this pyramid. The image is rescaled depending on the number of scales per octave (nPerOct). An octave is the necessary scaling to rescale the image by a factor of 2. So, each downsampled image is rescaled according to equation 1.1. The logic is the same for upsampling (equation 1.2) [Silva, 2013].

$$DownScale = 2^{\frac{-1}{nPerOct}} \tag{1.1}$$

$$UpScale = 2^{\frac{1}{nPerOct}} \tag{1.2}$$

Dollar et al. proposed a new approach for fast pedestrian detections, named "*FPDW*" [Dollar et al., 2010]. Instead of rescaling the input image N times, they propose to rescale it only $\frac{N}{K}$ times, see figure 1.8c. By reducing the number of image resizing and feature computations by a factor of K, the total detection time is significantly reduced.

Recently, Benenson et al. reversed the "*FPDW*" idea. They approximated the feature responses across scales and decided how to adjust a given stump classifier to classify correctly,

as if the feature response had been computed at a different scale.

Classification

There are multiple Machine Learning (ML) methods, each with its specifications and applications, so, one must choose a method that correctly fits the problem. The adopted method needs to classify between two classes, Pedestrian and Not Pedestrian, needs to handle thousands of features per sample and should also be fairly resistant to over-fitting. The method that best fits these requirements is AdaBoost, and a compact description will be carried out in the following section. The basic idea behind AdaBoost, short for Adaptive Boosting, is that it is possible to generate a very accurate prediction rule, or strong classifier, through the aggregation of rough and moderately inaccurate linear rules, weak classifiers, provided that they perform just slightly better than a random classifier would.A graphic illustration of this is shown in figure 1.6 [Silva, 2013].

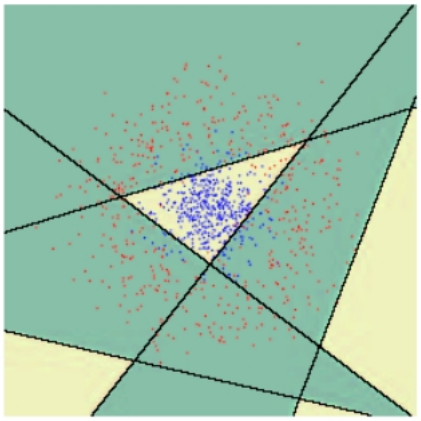


Figure 1.9: Weak classifiers aggregation

1.4.3 Sensor Fusion Information Systems

The LIDAR is used as a primary detection sensor generating ROIs which are projected onto the image. Multistage schemes can be arranged to combine information from LIDAR and camera in the same structure, in [Ludwig et al., 2011], a LIDAR-dependent processing stage is used to decrease the complexity and the processing time.

To deal with the problem of object scale variations in images, [Douillard et al., 2007], the classification method was evaluated and compared using several features: geometrical, visual and the combination of both. The authors used Conditional Random Field (CRF) and a LogitBoost classifier.

LIDAR and camera fusion can be performed using centralized or decentralized configuration [Premebida et al., 2009].

With the purpose to overcome the limitations of each sensor and to enhance the performance of the overall system, the LIDAR and image information is conceptually represented by two strategies:

- Centralized architecture: LIDAR and camera information are combined at the feature level, thus a common/central classification framework is used to process the laser and

the image-based features as a single, and jointly, feature vector.

- Decentralized architecture: information from each sensor is processed separately, hence the output of each sensor-based module, in the form of a likelihood or a confidence-score, is combined in the decision level.[Premebida, 2012]"

Broggi et al, approach a system designed to search designated areas (critical areas), in specific situation in urban scenarios (fig. 1.10)[Broggi et al., 2009]. By using both sensors, the two can scan the area if one cannot reach it. The main system characteristics are:

- quickly detect pedestrians, given the short working range and the particularly high danger of collision with a pedestrian suddenly appearing behind an obstacle;
- detect pedestrians as soon as they appear, even when they are still partly occluded;
- limit the search to specific areas, which are determined by a quick preprocessing;

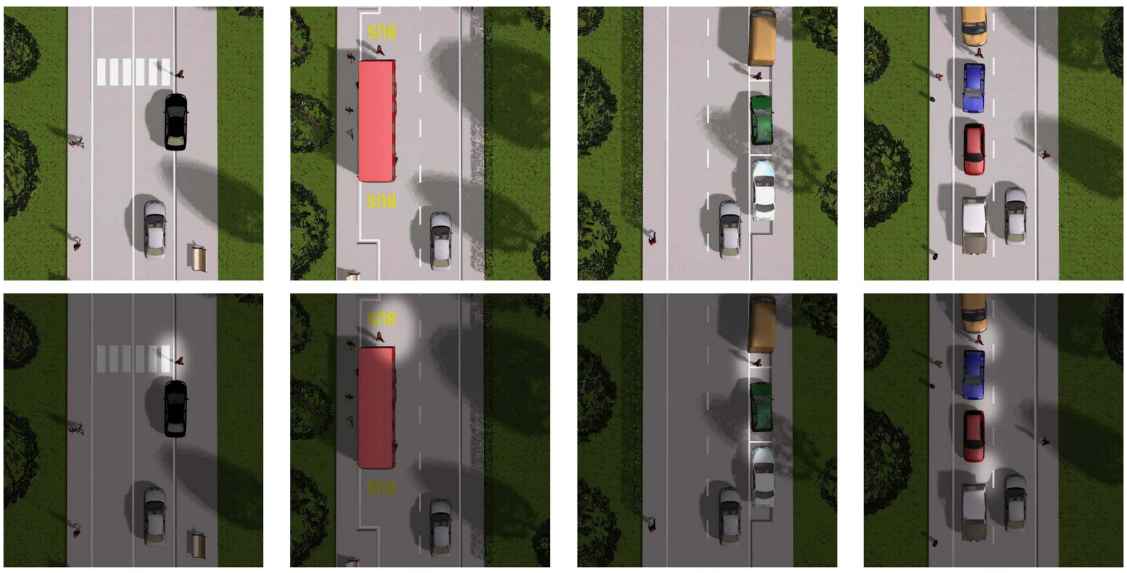


Figure 1.10: Shadow area locations[Broggi et al., 2009].

Fusion can provide a quick and robust detection in case of suddenly appearing pedestrians. A list of areas, in which a pedestrian may appear, are provided by the laser scanner, whereas the camera is able to detect the pedestrians, even when they are not yet visible to the laser scanner.

Pedestrians may not be detected by a laser scanner positioned in the front bumper but can be detected using vision, even if partially occluded, see figure 1.11[Broggi et al., 2009].

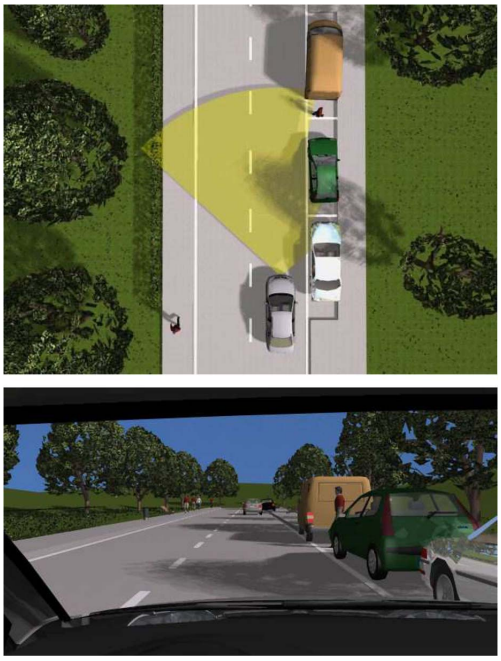


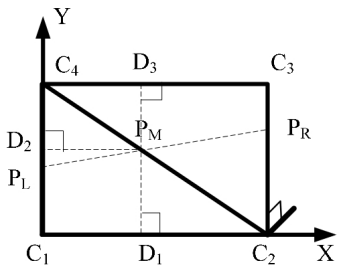
Figure 1.11: Pedestrian partially hidden by parked cars [Broggi et al., 2009].

1.4.4 Calibration Methods

The calibration is an essential way to determine the geometric transformation relationship between two coordinates, combining laser scanner and the image coordinates with the vehicle coordinates, so that in the end, the laser points world coordinates could be associated with image pixel points.

Li et al. designed a rectangular frame with one diagonal connected [Li et al., nd], shown in figure 1.12a. A bracket on the frame bottom corner was used to hold the rectangular frame perpendicular to the ground surface. This method allows calibration even when laser is not parallel to the ground, derived from its shape.

(a)



(b)

Figure 1.12: Calibration Tool Example: Rectangular frame with one diagonal connection [Li et al., nd]

Guan et al. developed a special right angled isosceles triangle board that allows automatic calibration [Guan et al., 2009]. Experimental tests on running vehicle showed that the system

could avoid interference, ambiguity and display the driving environment including obstacles, vehicles, pedestrians, figure 1.13.

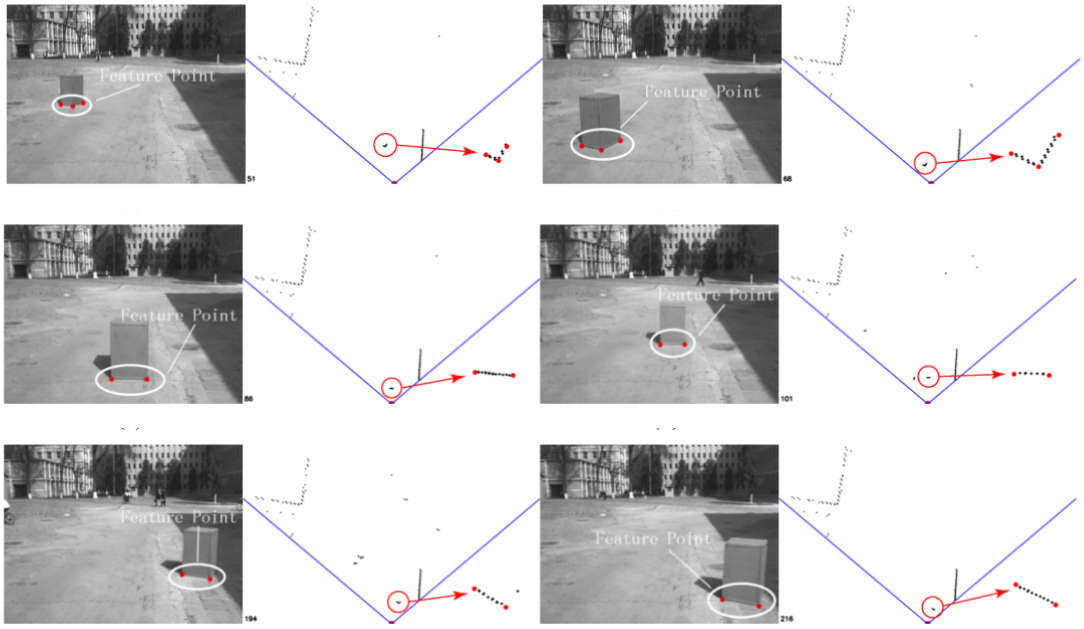


Figure 1.13: Feature points extraction and calibration processing [Guan et al., 2009].

1.4.5 Discussion

Despite existing some work in both subjects (vision pedestrian detection and pedestrian LIDAR-based tracking) outside LAR, it was decided to take advantage of the "Visual Pedestrian Detection using Integral Channels for ADAS", by Pedro Silva [Silva, 2013] and "Target tracking using laser range finder with occlusion", by Jorge Almeida [Almeida, 2010] algorithms.

It was used Pedro Silva algorithm, because full algorithm was provided.

Despite Jorge Almeida algorithm being recent (2010), it has already been cited in some master thesis. This demonstrates it's value. For this reason, this algorithm was used in this project.

Chapter 2

Experimental Setup

The programming language used was C++ under Linux platform and, in addition to this, a set of indispensable development tools were utilized; first section will focus on existing software tools and AtlasCar hardware, necessary for this project; second section will focus on used calibration methods, enabling the extraction of extrinsic camera parameters.

In this project two different types of algorithms were fused, an image pedestrian detection and a LIDAR target tracking, in order to detect pedestrians with a better performance than image pedestrian detection alone. Because the laser is invisible, laser points cannot be determined directly and they can only be reached indirectly, calibration is needed to know the location of sensors frameworks.

2.1 Software and Hardware Tools

2.1.1 Software

Robotic Operation System

The Robot Operating System (ROS) provides a flexible framework that is designed to create and maintain robot software. It has a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms, which helps to handle the output of different types of sensors, such as cameras, lasers, actuators, contacts and other common elements in robotics environments.

ROS allows the elaboration of an infrastructure that can communicate with any running process, establishing a communication between different software modules (nodes). This communication works in three steps: first, a node advertises a topic, then, once that topic is advertised, the same node is able to publish messages on that topic, and finally, those messages can be listened by any node that subscribes to that same topic. Such messages can be of any kind, from simple strings of characters, visual or laser data to special messages with several parameters. Figure 2.1 illustrates a very simple ROS communication architecture.

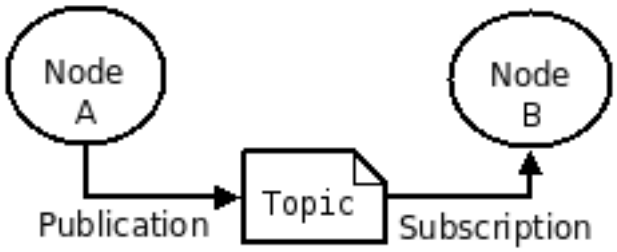


Figure 2.1: Simple ROS Scheme communication architecture

It is easy to understand that this structure for information exchanging has a great potential when applied to robotics since a uniform and standardized communication facilitates the development of complex multi-sensor applications. Another important feature in ROS is that it provides the possibility to record several messages (rosbags) that can be replayed later. This allows data to be collected in real scenarios, without worries, and be post-processed in a laboratory environment with the same conditions as those on the field. It also contains a visualizer, known as RViz, which is a very useful program because and in this case, it allowed the visualization of laser data, facilitating 3D points extraction for calibration process.

For these reasons, the work presented here was developed in ROS.

OpenCV

OpenCV is a popular open source computer vision library written in C, C++ and Phyton, that was designed for high computational efficiency and with a strong focus on real-time image processing applications. In this case OpenCV was used to extract extrinsic camera parameters, make image manipulation (threshold, draw rectangles, lines, labels), resize images, elaborate calculation, project laser points on image and write data on hard drive.

2.1.2 Hardware

Camera

First, for in-lab testing, a logitech c310 webcam was used, (figure 2.2a), with High Definition (HD) video capture at 720p (1280 x 720) with 30 Frames per second (FPS), perfect for steady footage but inappropriate when making a non-steady video, for example, with all the shaking in a moving vehicle, the result is a blurred video instead of a focused and clear image. This happens because the camera was developed for steady web video capture.

The car is equipped with a Bumblebee® XB3, (figure 2.2b), a 3-sensor multi-baseline IEEE-1394b stereo camera designed for improved flexibility and accuracy. The extended baseline and the high resolution (up to 1280 x 960) provide higher precision at longer ranges, having a maximum FPS of 16. It also allows the moving video to be smooth, making it appropriate to be the top camera in AtlasCar.

(a) Logitech HD c310 webcam [Logitech]

(b) Point Grey Bumblebee® XB3 [Bumblebee]

Figure 2.2: Hardware Cameras used

LIDAR

The Laser used was a Sick LMS 151, (figure 2.3) which is small and light-weight and has a maximum scanning range of 50 m, supply voltage range from 10.8 to 30 V DC, allowing the use of 12 V (car battery voltage) to power up the laser, a scanning frequency of 50Hz, and IP-67 protection class, enabling it to outdoor use.

Figure 2.3: Sick LMS 151 [Sick]

2.2 Calibration Methods

The calibration process allows the user to know the precise location of all the object frames from a reference frame perspective.

2.2.1 Manual calibration tool

Sensors on a tripod

In this phase, a manual calibration was tried to see how complex the process was and hopefully later on, try to calibrate the XB3 camera with the Sick lasers, with a similar procedure. Figure 2.4 illustrates the assembly made between LIDAR and the logitech webcam on a tripod, making it light, steady on every ground and easy to attach and detach. This allows gathering both data with fix translation and rotation between the two.

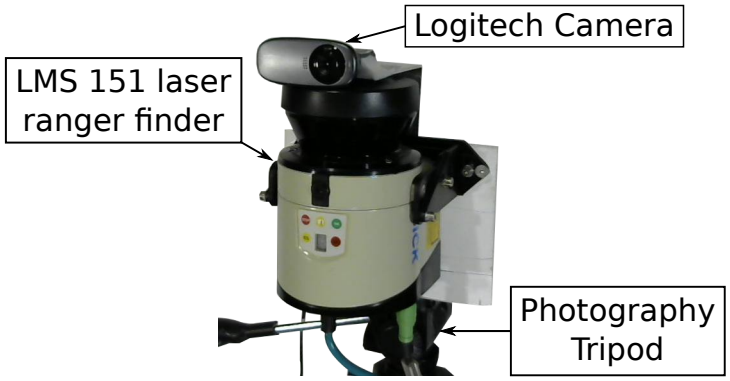


Figure 2.4: Sick LMS 151 and Logitech HD c310 webcam tripod assembly

The manual calibration for this set consists of 3 traffic cones displaced on the environment, (fig. 2.5) so that only the top tip of the cone is intersected by the laser, making it easy to extract the correspondent image points (px, py), but also the positions of the cones (x, y, z); because LIDAR data is a 2D laser-ranger finder, hence, z=0.

Once the images are being acquired by a camera, this sensor also needs to be calibrated.

Calibration is the estimation of a camera's intrinsic and lens-distortion parameters. Having the lens-distortion parameters, it was able to correct image distortion by transforming the image into a standard coordinate system. Matlab cameraCalibrator app was used to estimate camera intrinsic and lens distortion parameters, so that a rectified image could be obtained. The final output image is a rectified image.

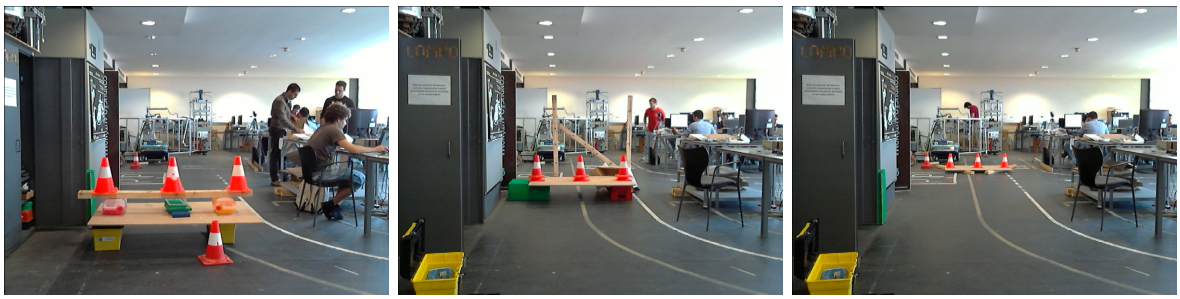


Figure 2.5: Manual Calibration Example: 3,8m / 6m / 10m from the LIDAR

After obtaining several image points (extracted from a rectified image) and object world points, they were saved in separate files.

For a rectified image, the distortion coefficient vector is considered null.

By using the OpenCV function, solvePnP, the object pose can be found from 3D (object point) and 2D (image point) point correspondences. Figure 2.6 illustrates how the function input and output works.

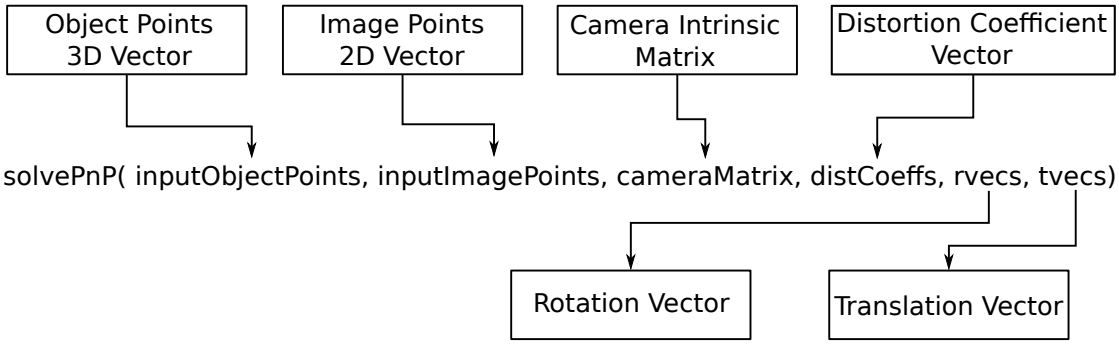


Figure 2.6: SolvePnP, OpenCV function (Inputs above / outputs below)

A rotation vector is the most convenient and compact representation of a rotation matrix (since any rotation has only 3 degrees of freedom); by using Euler–Rodrigues formula the vector can be converted into a matrix, for example using OpenCV Rodrigues function.

Having the rotation matrix and the translation vector, a [R|t] matrix can be created; this is a 4x4 matrix composed by a 3x3 rotation matrix and a translation vector; matrix 2.1 illustrates how the matrix is composed, being the r's the rotation matrix and t's the translation vector. To extract the position of the camera, from the LIDAR point of view, the ${}^{C}[R|t]_{FB}$ (Referential transformation of C (Camera) to FB(Front Bumper)) matrix needs to be inverted.

$$ {}^{C}[R|t]_{FB} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

Sensors onboard the car

After knowing what was necessary to calibrate a laser and a camera, the next phase was to calibrate the XB3 camera and the front left bumper Sick laser, Fig. 2.7.

The intrinsic parameters of the camera were calculated using [Caltech], whose intrinsic values obtained are summarized in Table 2.1.

Table 2.1: Intrinsic camera parameters: K

Focal Length	$fc : [1027.941; 1027.751]$
Principal Point	$cc : [658.264; 486.631]$
Distortion Coefficients	$kc : [-0.366; 0.206; 0.000060; -0.00016; -0.0784]$

The calibration was done by applying the same principles as described earlier, with traffic cones, modifying their height, so that only the tip could be intercept by the laser.

Figure 2.8 shows an example of the experimental calibration which is, how to extract the laser xyz coordinates and the correspondent image taken, to manually calibrate the LIDAR and XB3 camera.

Figure 2.7: XB3 and Front Bumper laser localization in the car

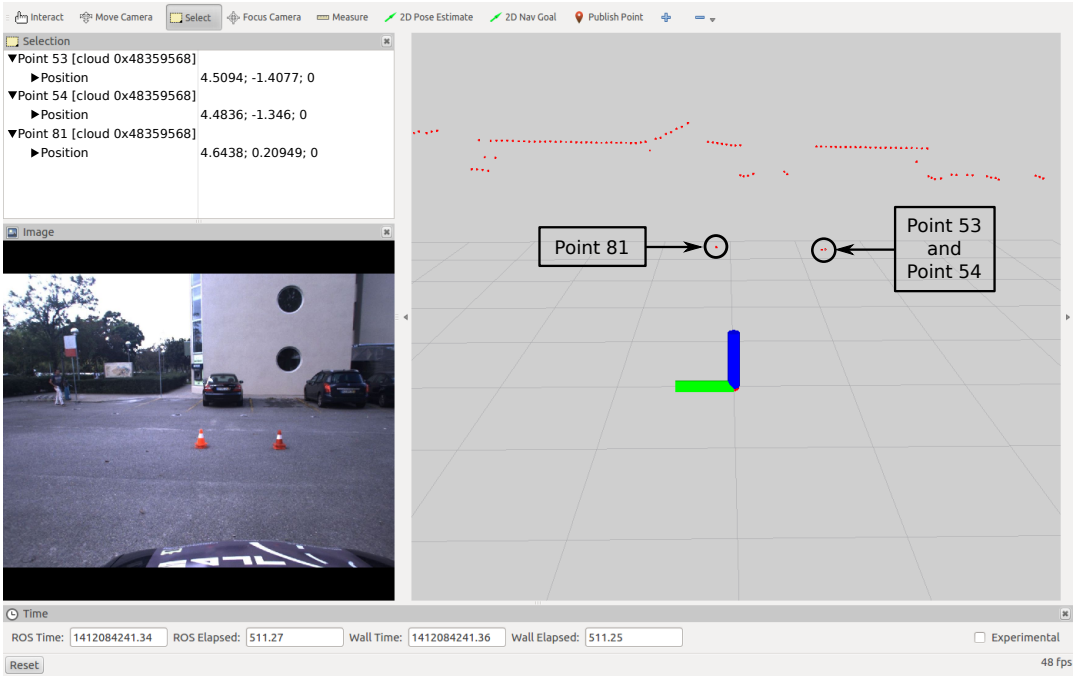


Figure 2.8: AtlasCar Manual Calibration Example

In some cases, multiple points correspond to one cone (figure 2.8), in these cases, a mean was made to sort out the correspondent middle point of the tip of the cone.

Using the object points, image points and the camera intrinsic matrix, the ${}^{C}[R|t]_{FB}$ matrix

could be obtained with the function described earlier (figure 2.6). The following matrix (2.2), necessary to make a rigid correspondence between the laser scanner and the camera reference system, was obtained, being the translation vector components in meters.

$$ {}^{C}[R|t]_{FB} = \begin{bmatrix} -0.0184 & -0.9998 & -0.0027 & 0.2337 \\ -0.1384 & 0.0052 & -0.9904 & 0.8455 \\ 0.9902 & -0.0179 & -0.1384 & 1.9781 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2} $$

Knowing these parameters, the location of the camera can be virtually shown in Rviz (figure 2.9).

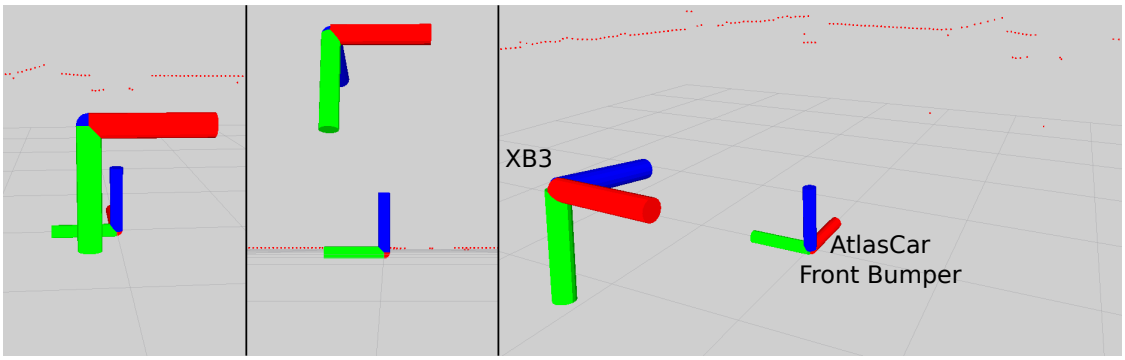


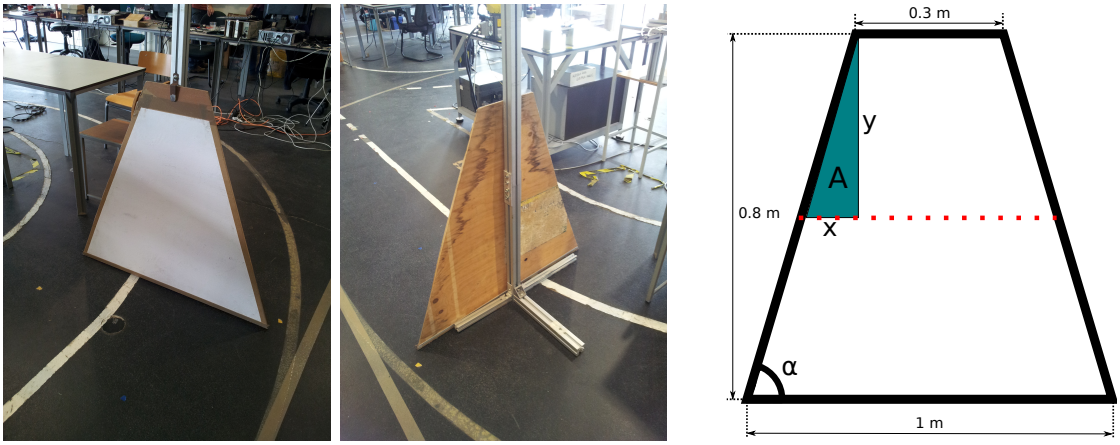
Figure 2.9: Camera view, laser view and XB3 location frame in world frame

2.2.2 Automatic calibration tool

Despite not being able to make a successfully automatic calibration tool, an algorithm was developed in order to provide it. The object shape used was a trapezoid (figure 2.10a); it has the following parameters: 1m max width, 0.2m min width, 0.8 height. Figure 2.10b shows the laser point extraction process, where the red dotted line illustrates a possible laser read and the "A" area, the consequent used values to extract the percentage height to withdraw points on the image.

The region was painted white, with the edges in brown tape, to easily distinguish the object from the environment.

The idea was to set the laser parallel to the floor, reducing inclination problems, thus enabling extraction of parallel points on the real image.



(a) (front on the left / back support on the right)

(b) Point extraction illustration

Figure 2.10: Object used to automatize the calibration process

The initial object angle (α) is 66.37. To withdraw the object width through laser data, "$laserwidth$", "y" can be calculated through equations 2.3 and 2.4. To transfer laser position ("y") to image coordinates, the value was converted to percentage. This allowed that the laser location could be extrapolated, despite the size of the object, equation 2.5.

$$x = \frac{laserwidth - 0.3}{2} \tag{2.3}$$

$$tan(\alpha) = \frac{y}{x} \tag{2.4}$$

$$\%laserheighttotop = \frac{y}{0.8} \tag{2.5}$$

After calculating the laser position, the algorithm grabs an image and scans it, looking for the white board position and extracting its height in pixel, then the $\%laserheighttotop$ variable is used to choose the pixel line where the laser intercepts with the white board. Where "B" stands for Board, the maths made is:

$$ImageLaserLocation = (BBottomPixel - BTopPixel) \times \%laserheighttotop \tag{2.6}$$

Knowing the vertical pixel where the laser intercepts the board, and the location of the board to make it stand out, the edge pixel can be obtained. These points, image (2D) and laser (3D), are used with the same principle as previous manual calibrations, to extract extrinsic camera parameters.

Figure 2.11 shows that this calibration method was not reliable. In this process, the calibration board needs to be completely parallel to the car, so that a perfect point extraction can be obtained. Not having a perfect board displacement, the extrinsic camera matrix was obtained with errors.

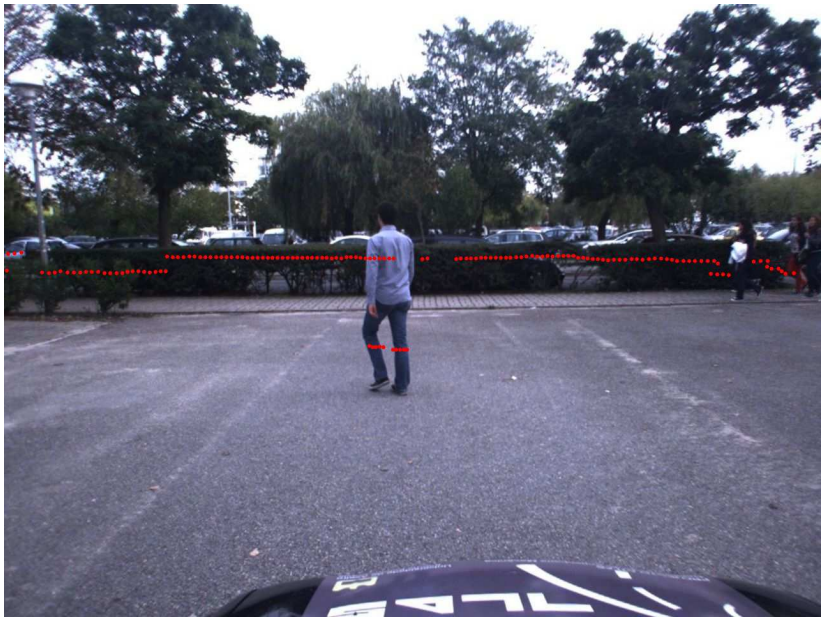


Figure 2.11: Automatic Calibration Result Image

2.2.3 Discussion

Comparing the previous results of both (manual and automatic) *AtlasCar* calibration methods, the "Manual" was the chosen one to continue this work. The transformation matrix used on the results above, was ${}^{FB}[R|t]_C$.

In table 2.2, the rotation matrix (3x3) was converted to RPY angles for a better comparison; "Automatic" calibration method has a huge difference in translation when compared to the "Manual" method. The *AtlasCar* system already had a calibration, but to prevent possible twisting in camera and lasers, the calibration was made again. "Manual" and "Existing" values are very similar, leading us to believe that the tests were correct.

Table 2.2: Calibration results (Manual vs Automatic)

	RPY angles			Translation		
	Roll	**Pitch**	**Yaw**	**X**	**Y**	**Z**
Existing	-1.7104	-0.0000	-1.5708	-1.8405	0.2400	1.1170
Manual	-1.7097	0.0027	-1.5892	-1.8375	0.2646	1.1118
Automatic	-1.7709	-0.0059	-1.5769	-2.5894	0.2380	1.8401

Chapter 3

Algorithm for Active Pedestrian Detection

This chapter describes the different stages needed to achieve Active Pedestrian Detection (APD); figure 3.1 summarizes the process. Pedestrian protection systems can be divided, in general words, in two fields of research: passive and active safety systems.

Active safety systems, of a high interest here, are based on pedestrian detection using sensors onboard the vehicle, and/or on the infrastructure, with the role of predicting and anticipating possible risks of collision [Gandhi and Trivedi, 2007].

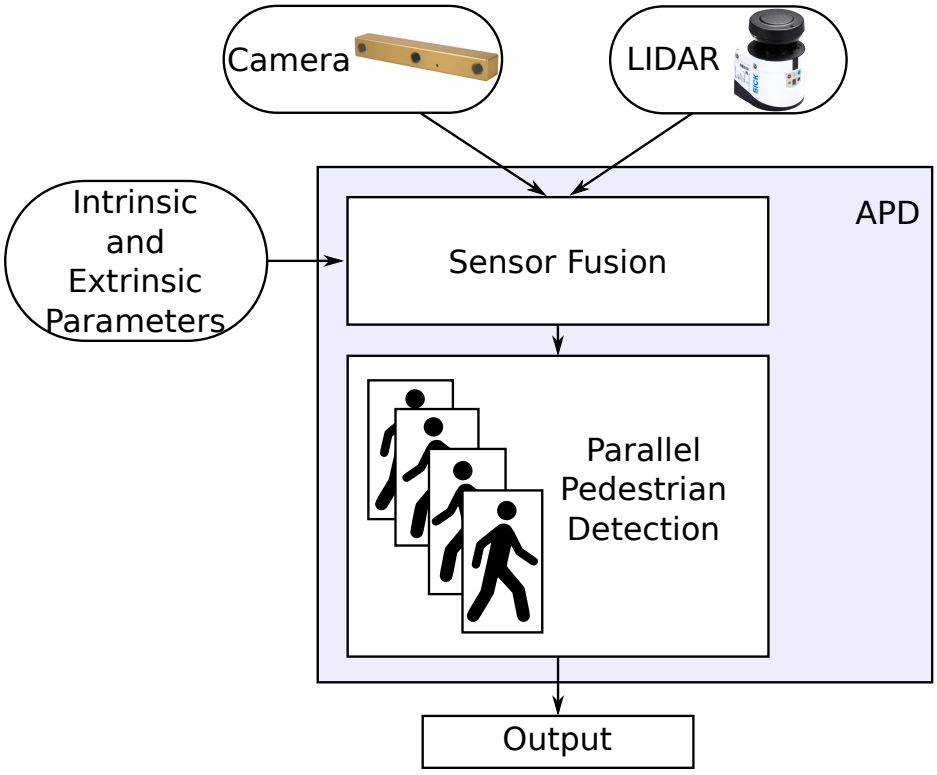


Figure 3.1: APD resume

3.1 Re-Parametrization of Existing Pedestrian Detection Algorithm

The original pedestrian detection algorithm, by Pedro Silva, subscribes an image published on ROS environment and resizes it to 640x480 for optimal processing [Silva, 2013].

The program had to be adapted to work with images of any sizes.

The previous detection process of scanning pedestrians on an image is shown on the left side of figure 3.2.

When interpreting the former process, one stage of the diagram can be removed, because when combining LIDAR and Camera information, the output is a smaller one-person image, with known size.

The algorithm uses the dense image pyramid process (figure 1.8b) for pedestrian detection. Because all the algorithm architecture was built around it, the process cannot be switched, but some topics can be modified to work with one-person images.

The dense image pyramid process rescales the image multiple times, while the model keeps intact. Knowing this, the given image needs to be rescaled, for correct pedestrian detection (in this type of process). Given a one-person image to classify, the algorithm does not need to spend too much time scaling the entire area of the initial image (640x480). Instead of that, a smaller image runs multiple times to make sure a pedestrian is well classified. Changing input image sizes, the algorithm running time decreases, allowing a quicker classification.

Understanding that the given image already has a potential pedestrian, the step search was decreased, allowing the image to be swept with more detail. The step search consists of the number of pixels the DW skips, when making an image column scanning- The process continues the same along the lines.

The output result was also modified. After ending the image scan, the original algorithm draws rectangles on different image locations. Now, after ending the potential pedestrian scan, the only output is the result of a single classification, (Ped or nPed).

The modifications made can be visualized on the right side of figure 3.2.

3.2 Sensor fusion algorithm

This section describes the process to separate potential pedestrian from the rest of the objects and the development of an application for pedestrian active detection and its explanation.

3.2.1 LIDAR Classification of Potential Pedestrian

After knowing where the objects are, in a LIDAR scan, the potential pedestrians need to be organized to be further classified by a more robust algorithm. By applying the Multi Target Tracking (MTT) algorithm[Almeida, 2010], IDs can be attributed to different groups of laser points, allowing the objects to be tracked continuously, until they vanish from the laser radar.

Initially, only the object width was used to sort out the potential pedestrians from the LIDAR data. Figure 3.3 shows that only with this condition the number of potential pedestrians was narrowed to a few, compared to the amount of the objects found by the LIDAR. The blue rectangle shows the origin of the *AtlasCar* coordinate frame, the black labels correspond to object IDs.

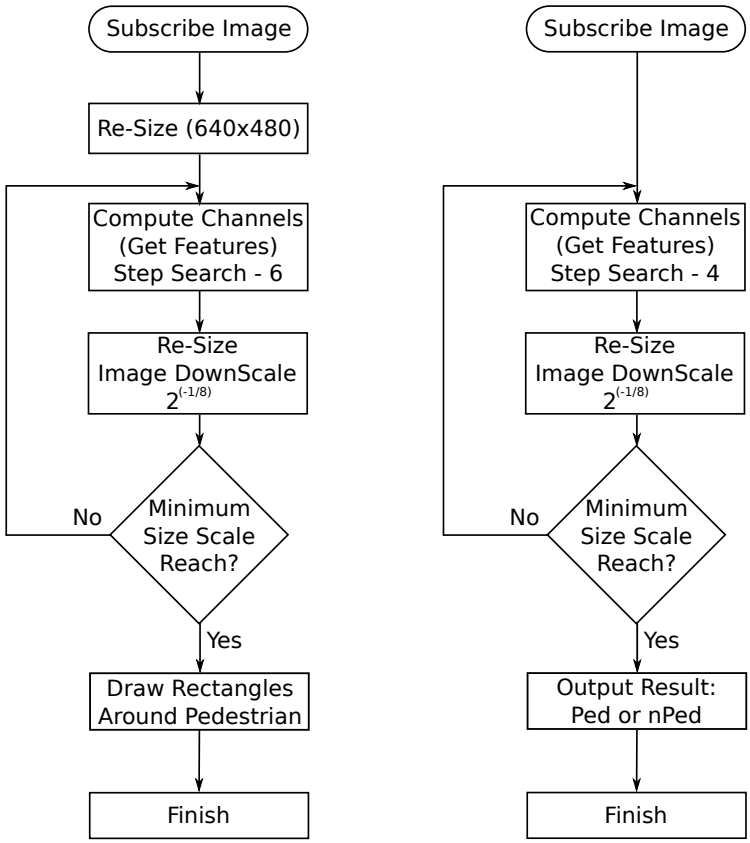


Figure 3.2: Pedestrian Detection Algorithm Scheme (before on left/ after on the right)

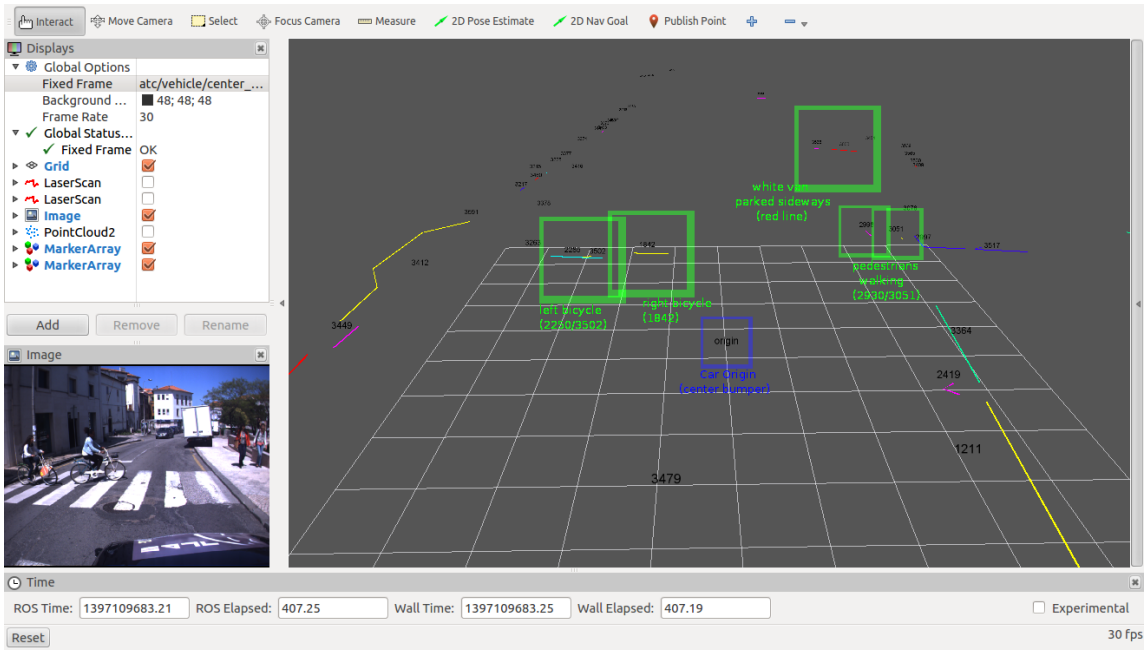


Figure 3.3: Potential Pedestrian Search Example shown in RViz, with the correspondent image on the lower left

3.2.2 Active Pedestrian Detection

The Active Pedestrian Detection algorithm consists of 4 stages: candidate list creation, candidate classification, candidate visualization and candidate elimination. These stages are explained below in the same order.

The application starts subscribing the MTT data and the camera color rectified image at the same time. The image is only used after the MTT data is received and processed. Until that moment the image is only shown, as if the application is a mere camera info and a rectified image reader.

Candidate List Creation

After making a fast-low resolution classification with the LIDAR data, the candidates are then stored with their properties: ID, distance to LIDAR in a straight line, left and right world point of the candidate, correspondent image in a vector and classification of the candidate. The ID is the only fixed value, until the candidate is removed.

The candidates vector is cleaned before starting the program.

Figure 3.4 illustrates the LIDAR candidate list creation process, when new LIDAR data is subscribed.

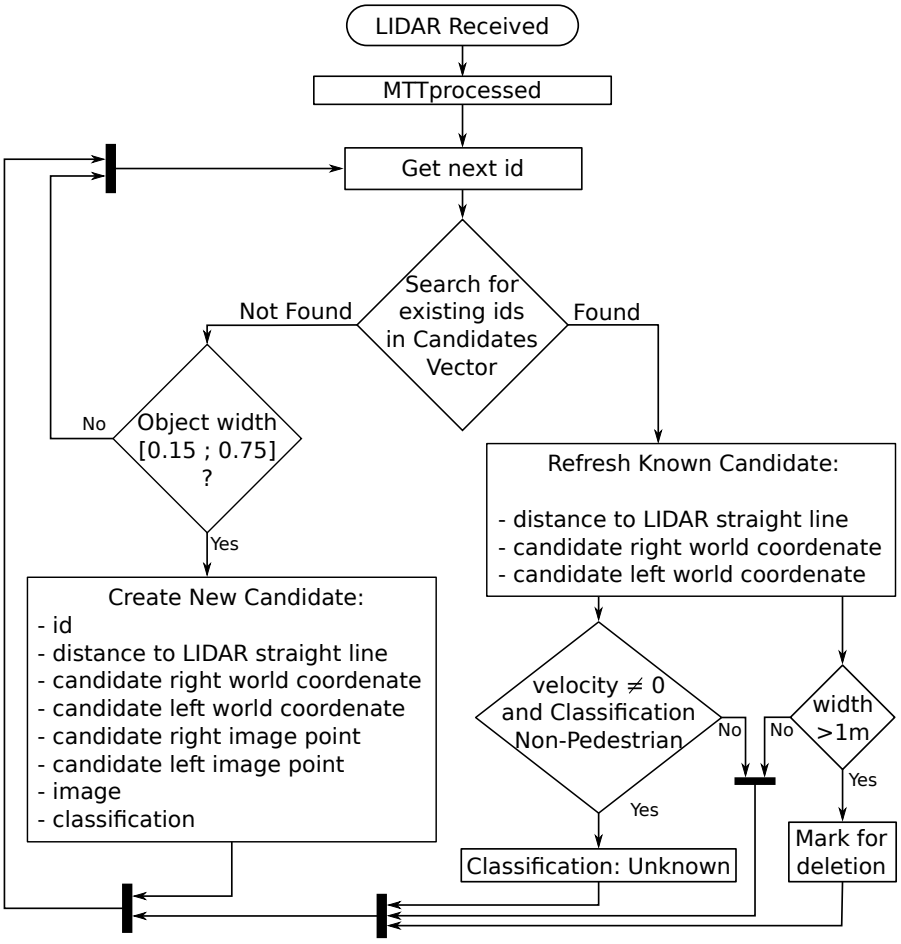


Figure 3.4: Sensor Fusion LIDAR Process Scheme

After the MTT data subscribed, the search for the ID in the stored vector and the Target data begins. Initially the candidates vector is null, therefore, there is a need to create a new candidate.

For the MTT target to be considered a potential pedestrian, the object needs to have a width considered in this range [0.15;0.75] (m).

After scanning all the incoming Target data, the candidate vector is sorted by distance to LIDAR in a straight line, so that the candidate closer to the LIDAR can be classified first, for safety reasons.

After receiving multiple times MTT data, IDs start to match the ones in the candidates vector. After being found, the candidate vector is refreshed. Candidate location is updated.

If the candidate classification is Non-Pedestrian and its velocity is non zero, the classification changes to "Unknown". Normally, an object does not have motion. This fact helps the candidate classification process to reclassify and make sure the object is not a "Pedestrian".

If an object has a width larger than 1 meter, the candidate is marked for deletion. This parameter was tested and no pedestrian was ruled out. For example, if one candidate walks along a wall, the wall can be added as a new candidate. However, the candidate is walking off, the width of the wall starts to grow. If there had not been conditions to control this factor, the wall could be marked continuously. This problem is illustrated in figure 3.5. Initially the pedestrian number 19767 walk left to right on the sidewalk, making object number 19662 appear. This was corrected using the "$>$ 1 meter" parameter.

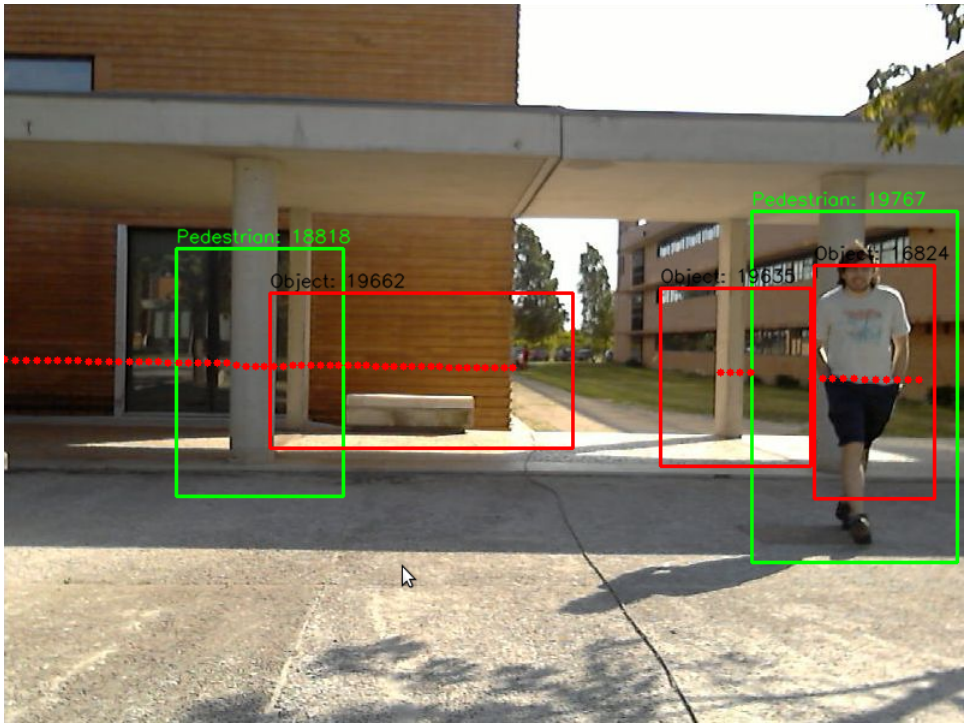


Figure 3.5: Sensor Fusion LIDAR-Camera Scheme

Candidate Classification

When finished sorting the candidates vector, the classification process checks if there are any candidates available (figure 3.6); if so, by knowing the location of the candidate on the image, one portion is cropped and attached to the candidate image topic, in the candidate vector; if the candidate already has a classification different from "Unknown", it means that it already has an image of itself, so the extract and classification process is skipped, the same is done until candidates run out.

Classification process runs every time an image is received.

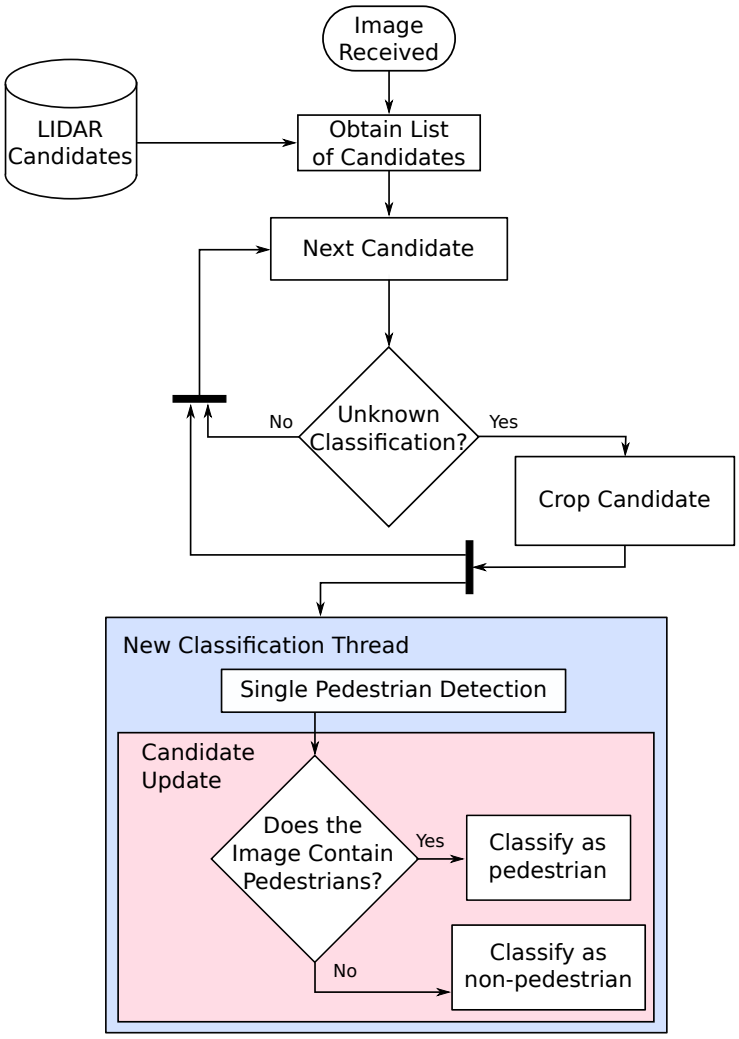


Figure 3.6: APD candidate classification process

Afterwards, a thread creation enables the scan to continue, instead of spending time waiting for the verdict. The multi-thread process helps the program to classify different candidates at the same time, running the classifications alongside the application.

The thread calls the Single Pedestrian Detection algorithm, giving it the candidate scaled image for classification. When finished, the candidate classification status is changed to Pedestrian or Non-Pedestrian depending on the verdict of the algorithm.

Candidate classification is changed to "Processing" until classification thread terminates, see figure 3.8 to acknowledge the different types of visualized boxes.

Candidate Elimination

While the candidate remains in the LIDAR sight, the candidate vector continues to be updated. When the candidate walks out of the LIDAR sight zone, its vector is eliminated. This process needs to occur, or the candidates vector would grow continuously, until stopping the program due to memory leak.

Another reason for elimination to occur, is that the classification can now focus on classifying the objects that can be a real threat.

The most complicated case for elimination, is when a candidate is still being processed by the classification thread and has already left the LIDAR sight zone. In this case, the thread is forced to shut down, to enable candidate elimination without damaging the candidates vector.

Elimination process is needed, because it allows a better candidates vector management and increases the overall algorithm performance. Figure 3.7 summarizes all the elimination process.

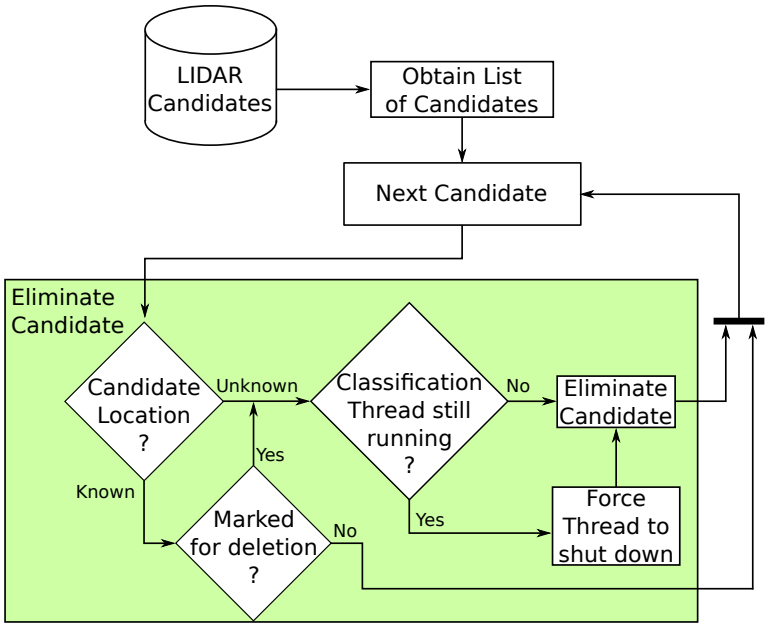


Figure 3.7: APD candidate visualization process

Candidate Visualization

The candidate visualization only is needed for: debug process, driver assistance and a qualitative visualization of the classification results.

For autonomous driving, this part is not important, because no driver is onboard to visualize the classification, persisting the essential topics (3 previous ones).

The candidate is classified while remaining in the image region. For different classifications, different visuals are needed, so it was proposed distinct color rectangles (Bounding Box (BB)), to highlight classification. Whether the candidate has the Pedestrian or Non-Pedestrian status,

this is highlighted on the seen image with a green or a red BB, respectively. The "Unknown", is also highlighted with a red BB, see figure 3.8.

Figure 3.8: Different types of Candidate classifications, Not Processed (Unknown), Processing, Pedestrian, Object (Non-Pedestrian), left to right respectively.

Chapter 4

Experiments and Results

To test the application, experiments were made in three different places: the first one inside the LAR, then the tripod assembly facing the Mechanical Engineer building and finally with the *AtlasCar* facing a sidewalk. The test was moved to the outside in order to check if it had a good performance while some people were walking side by side and crossing each other.

To have quantitative results one needs to create a ground truth. It defines the ideal output of a perfect detection. Therefore, it was obtained by manual selection. Actually, every single algorithm output image needs to have a correspondent ground truth image, to enable correct result extraction.

Classification performance metrics

The classification performance considered on this test and the principles used to classify TP, TN, FP, FN, are explained in Table 4.1 [UniversityofRegina].

Table 4.1: Terminology and definitions for performance metrics

	Ground truth	Algorithm Output	
True Positive (TP)	Present	Found	equiv. to hit
True Negative (TN)	Not Present	Not Found	equiv. to correct rejection
False Positive (FP)	Not Present	Found	equiv. to false alarm
False Negative (FN)	Present	Not Found	equiv. to miss
True Positive Rate (TPR)			TPR = TP/(TP + FN)
True Negative Rate (TNR)			TNR = TN/(TP + FN)
False Positive Rate (FPR)			FPR = FP/(FP + TN)
False Negative Rate (FNR)			FNR = FN/(FN + TP)

Figure 4.1 displays the different types of classification, as described in Table 4.1, while figure 4.2 shows the ground truth where the known real pedestrians are highlighted for comparison.

The list below describes each BB, drawn in figure 4.1, left to right. Each topic explains the BB classification, comparing it to the ground truth image, figure 4.2:

BB 1. A column highlighted by a green BB: it means that a pedestrian has been detected in

that location, but when compared to the ground truth image, no pedestrian is present in that location, making it a FP classification;

BB 2. The first red BB means that no pedestrian was detected; comparing it to the ground truth image, no pedestrians had been detected on both, so the output classification is TN;

BB 3. The pedestrian is correctly highlighted in both images, classification TP;

BB 4-5. Red BB resembles the same condition as BB 2., output classification TN;

BB 6. Last BB is red, so outcome is "Not Found". On the other hand, the ground truth has a green BB, meaning "Pedestrian Present", making the output classification FN;

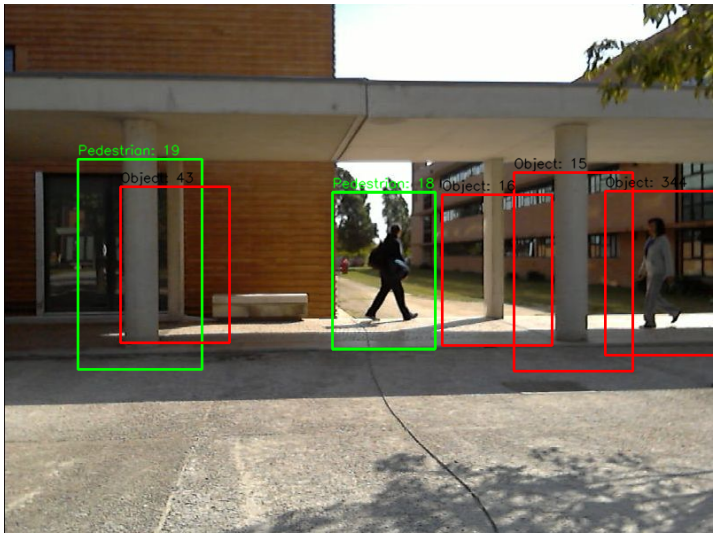


Figure 4.1: Example of output image and results of a Pedestrian classification

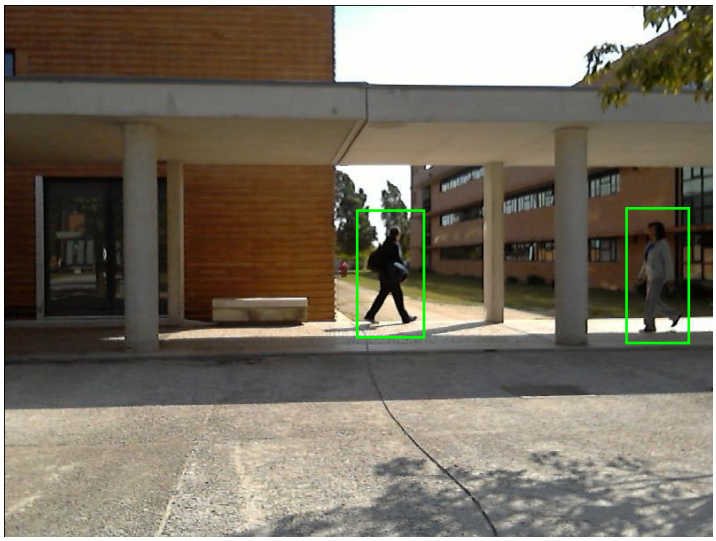


Figure 4.2: Ground truth for the same image as in figure 4.1

4.1 Experiment 1 - Tripod mounting indoors

Actually, in spite of not being big, the LAR was very helpful at testing if the candidates classification and tracking were fine. By using the assembly in figure 2.4 some data was extracted.

The first time the application was processed, non candidates were being selected, this resulted in an image filled with red BBs. Each BB corresponds to a LIDAR candidate (not selected), see figure 4.3.

The red dots seen in figures 4.3, 4.4 and 4.5, approximately in the center of the image, are the projected laser points.

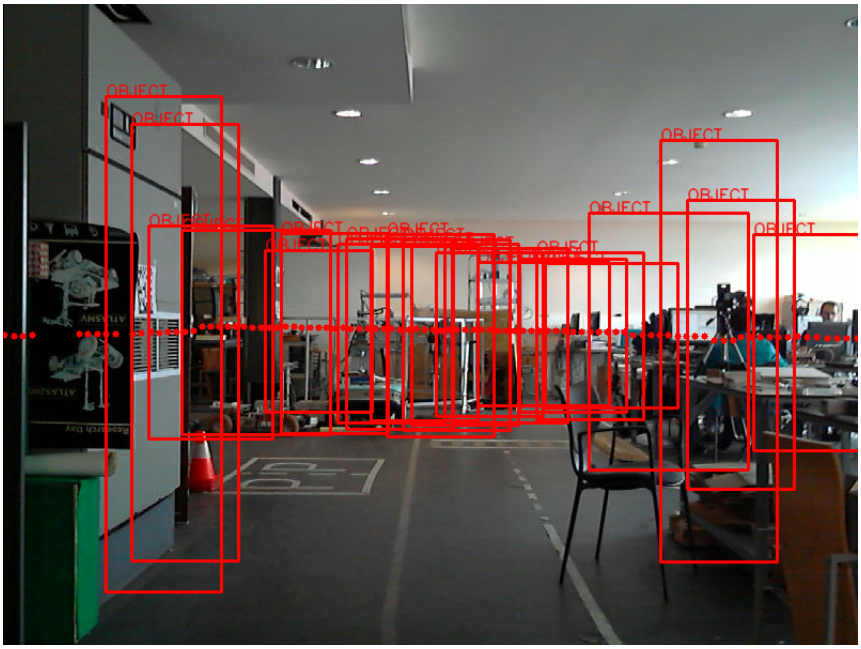


Figure 4.3: LIDAR objects shown, without any previous selection

After running all 4 stages of the program, the massive amount of BB disappeared, remaining only Potential pedestrians for classification, see figure 4.4. Some of the candidates are still being processed (status: PROCESSING). At the same time, the algorithm has already processed some objects and one pedestrian.

Observing that the program was running as expected (not excluding any pedestrians from the LIDAR search), some modifications were made concerning the visualization tool.

The font color was changed to black as well as the font type of the status on top of the rectangles, when the candidates are not pedestrians. Candidate ID was added to the status, making it easy to extract the time of classification in the terminal, for the initial debug. See changes in figure 4.5.

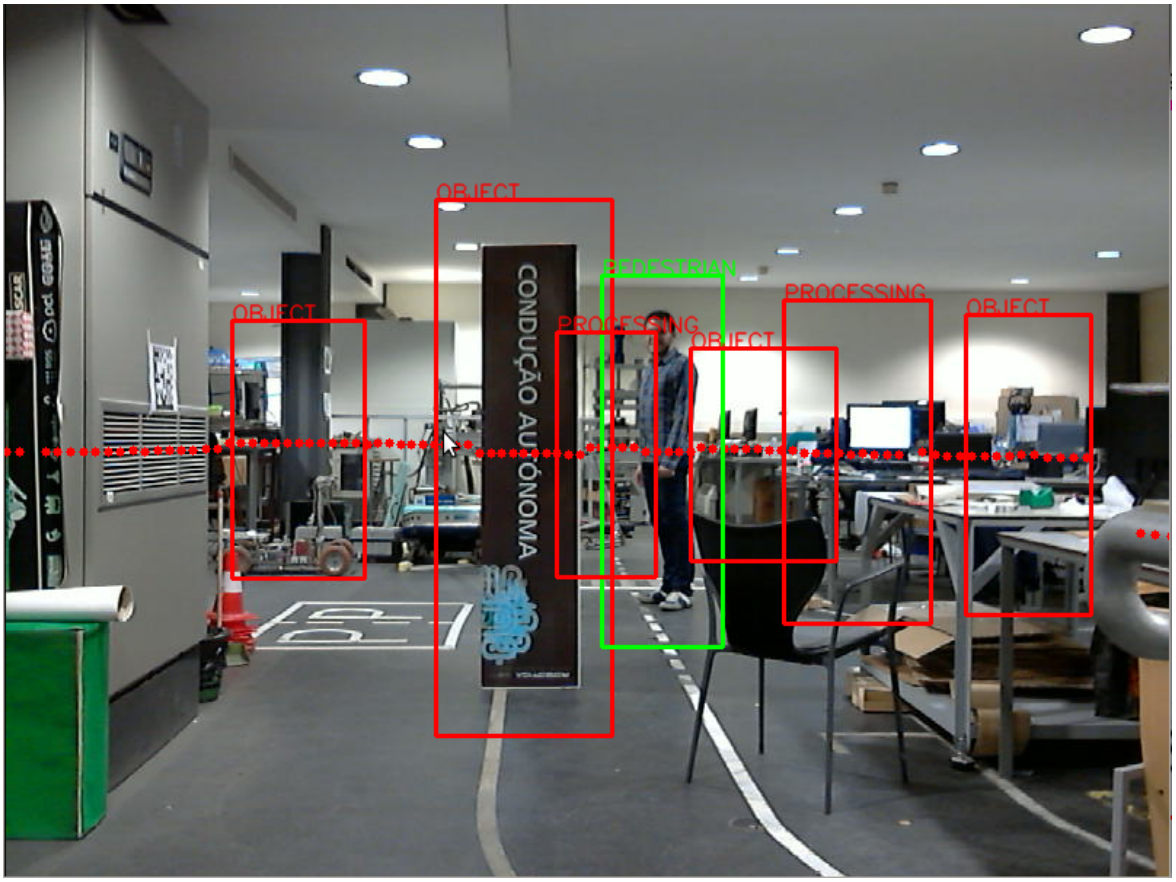


Figure 4.4: Differences between candidates, Object/Pedestrian/Processing

The candidate time of classification is directly related to LIDAR distance. The farther they were from the LIDAR, the faster they were to classify, because the candidate image gets smaller.

One particular case happens when a person, instead of walking towards the LIDAR, walks in the opposite direction. This lead to a much longer classification processing time than it was used to.

Bearing in mind these results, a modification had to be made. Actually, before sending a candidate image to the Single Pedestrian Detection Algorithm, depending on the actual size of the image, a rescale had to be made. For example, if the image has a pixel height superior to 250, the image is rescaled to 250 pixels. The image rate is preserved.

The rescaling process allows candidates which are close to the LIDAR, to be classified faster than they used to be.

Figure 4.5, shows the last modifications made, the different font type, color and added IDs. On average, a candidate image this size (Pedestrian: 189) would take 10 seconds to classify; with the rescale modification the classification only took 3.27 seconds.

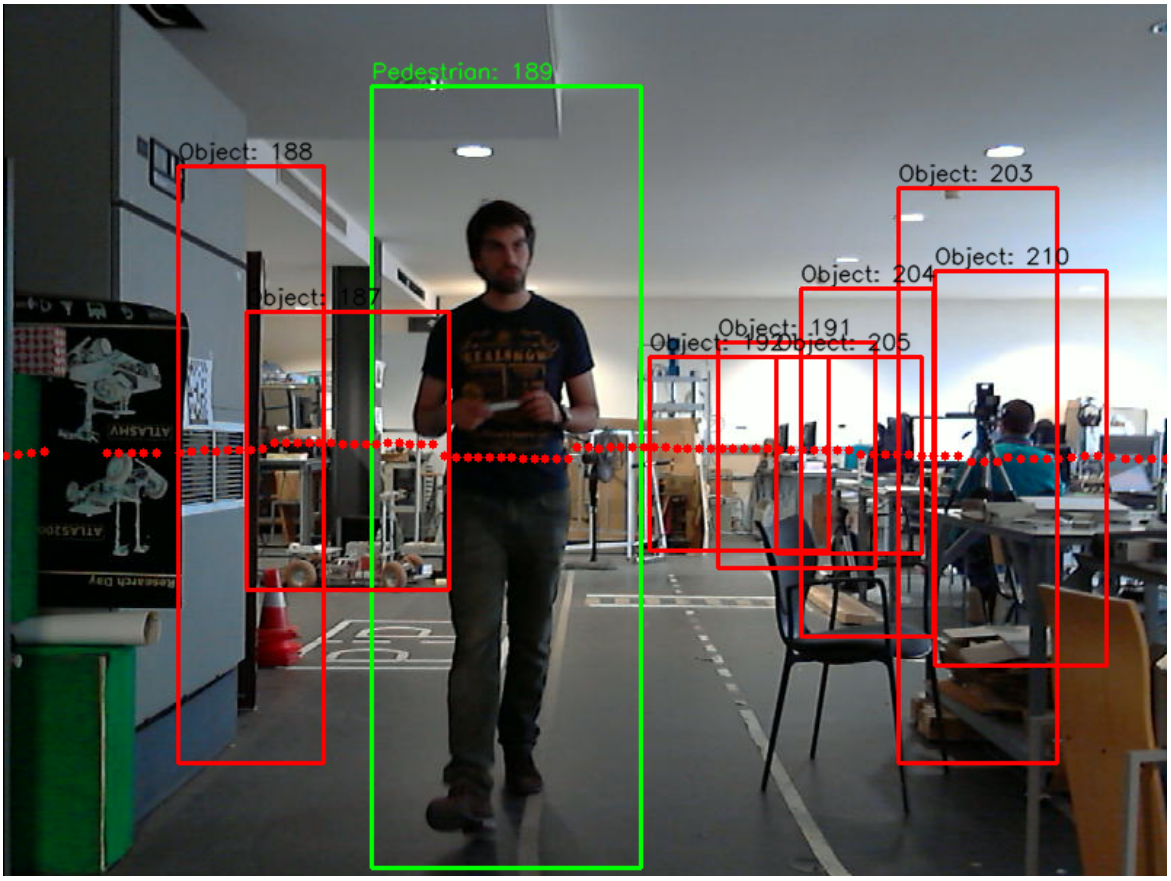


Figure 4.5: Figure shows last modification made to the algorithm

4.2 Experiment 2 - Tripod mounting outdoors

The next step was to take the assembly outdoors. The exterior of the Mechanical Department sidewalk seemed to be the most appropriate spot to carry on this experiment, as here there are many pedestrians at any time of the day. In fact, it became suitable for gathering pedestrians data. (see tripod mounting location in figure 4.6.)

The gathered data allowed the comparison between the two algorithms (Sensor fusion and Pedestrian Detection).

Furthermore, a comparison requires the existence of a ground truth data. Therefore, the comparison was made by using a manual labeling of the pedestrians positions in the image.

The data gathered for this experiment had 2275 image files. Because the logitech camera, figure 2.2a, records at a rate of 30 fps (frames per second) and to avoid catching the pedestrian always on the same spot, the 2275 images were reduced to 220, containing 127 ground truth positive bounding boxes.

The results shown in figure 4.7 were obtained (compared with the correspondent ground truth) considering the Pascal criteria, where two BBs are considered a match if the overlap area is superior to 0.5, [Everingham et al., 2009].

Red lines represent the results only with the Pedestrian Detection Algorithm, whilst blue

lines correspond to Sensor Fusion Algorithm results.

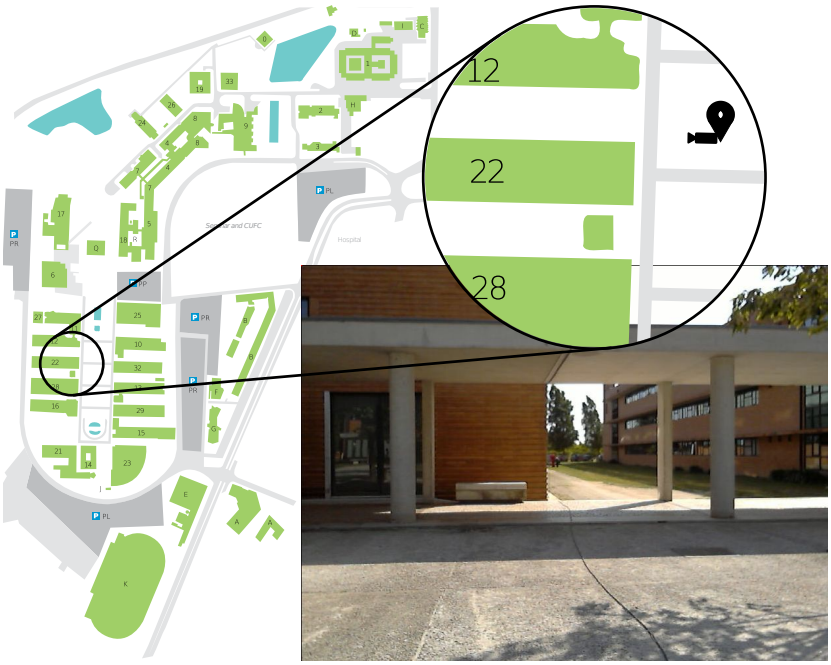


Figure 4.6: Tripod Assembly Spot Location on the UA Campus, 22 - Mechanical Engineering Department [UA]

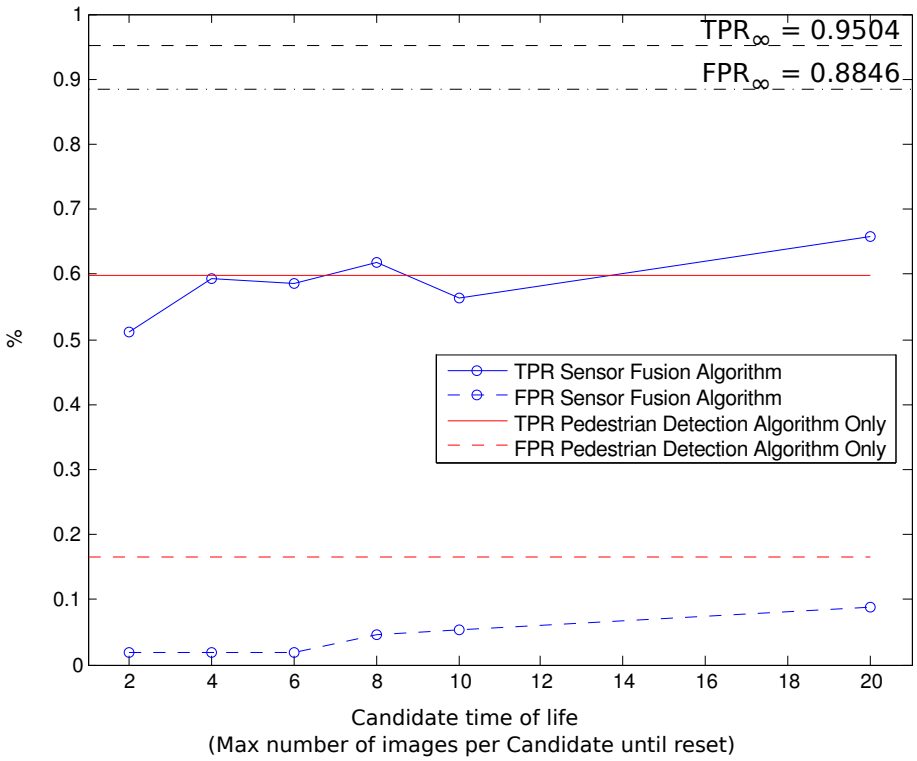


Figure 4.7: Experiment 2, TPR (True Positive Rate) and FPR (False Positive Rate) Results

Table 4.2: Experiment 2 detailed results

Life time	Sensor Fusion Algorithm		Pedestrian Detection Algorithm	
	TPR	**FPR**	**TPR**	**FPR**
2	0.5116	0.0185	0.5982	0.1653
4	0.5938	0.0187		
6	0.5852	0.0189		
8	0.6179	0.0450		
10	0.5625	0.0541		
20	0.6589	0.0885		
∞	0.9504	0.8846		

The Sensor Fusion Algorithm has some parameters that can be changed. The one that makes the biggest difference, is the time of life of the candidates vector. This way, further research was made through different tests, changing the time of life of the candidate.

On these grounds, the time of life of the candidate was counted through the number of images passed with the same classification. The candidates vector is reset every time the image counter equals the number of the given life time. For example, if the life time is "2", candidates classification is reset every two images received.

Based on this evidence, one can say that resetting the candidates vector means that every classification is changed to "Unknown". Further tests were carried out with 2, 4, 6, 8, 10, 20 and ∞ ($\infty \rightarrow$ there wasn't a candidate time limit). The Pedestrian Detection Algorithm was run with the same parameters as the Sensor Fusion Algorithm, to allow comparison (Step search = 4).

To provide a better understanding of how the fusion detection works for the "∞" case and why the FPR is high compared to other experiments, some example results are shown in figures 4.8 through 4.11, in which the program itself drew green and red BB automatically. Figure 4.12 shows an example of FPR for the Pedestrian Detection Algorithm, an object that was far away and was still highlighted. This can be easily avoided with the help of the LIDAR.

The data also allowed a process time comparison, see table 4.3. The process time tends to decrease, when candidate time limit increases.

Table 4.3: Mean time (in seconds) to process an image

Life time	Sensor Fusion Algorithm	Pedestrian Detection Algorithm
2	0.4021	61.5298
4	0.3766	
6	0.3648	
8	0.3539	
10	0.3973	
20	0.2905	
∞	0.1478	

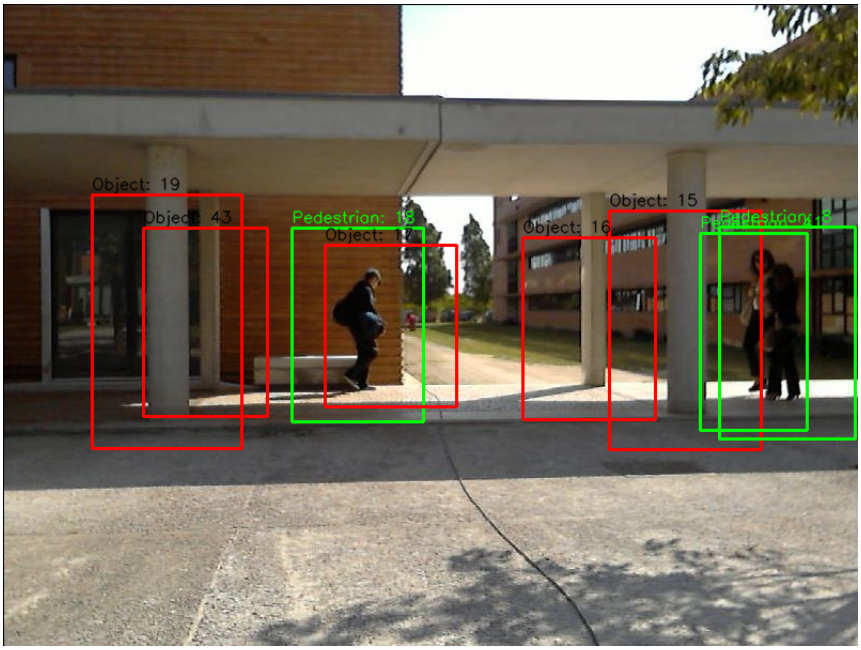


Figure 4.8: Example Result 1 - TP and TN classifications.

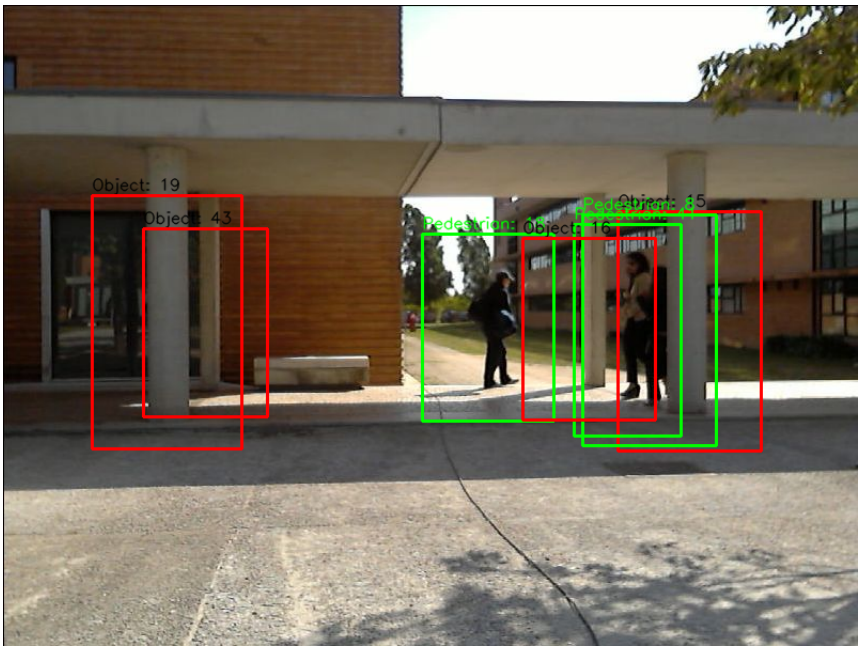


Figure 4.9: Example Result 2 - Target tracking allows Pedestrians to maintain the same ID, even when occluded by a column.

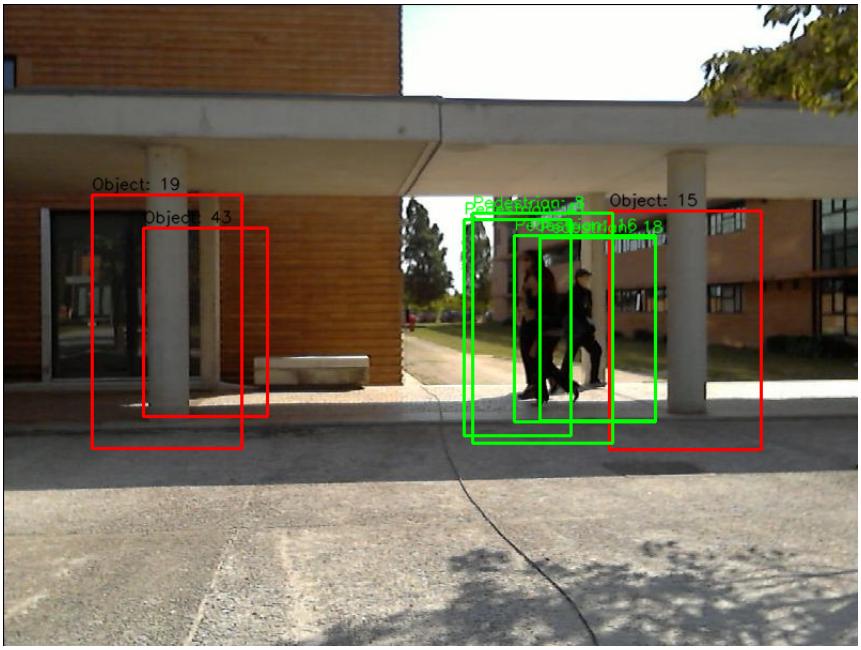


Figure 4.10: Example Result 3 - Evidence shows that the tracking gets confused with some people crossing (3 pedestrians and one column -4 objects in total-). Moreover, it appears to be very complex for the further objects to be correctly tracked, since the surrounding BB of the column has now a pedestrian ahead, which induces into a classification error, switching it from "Object" to "Pedestrian", figure 4.11. The nearest pedestrians continued to be well tracked.

Figure 4.11: Example Result 4 - Outcome of fig. 4.10, previous Pedestrian 18 became Pedestrian 46, reclassified after receiving a new ID.

Figure 4.12: FP example using Pedestrian Detection Algorithm

4.3 Experiment 3 - AtlasCar

As experiment 2 provided some satisfactory results, some data was extracted aboard the *AtlasCar* in order to make evidence more realistic and reliable. This way, the choice of a location near the *AtlasCar* garage, figure 4.13, seemed adequate for this purpose (test the algorithm), since it is the university students ordinary footpath to attend classes and is, in fact, crowded all day long.

Previous parameters were maintained while conducting this experiment.

The non-existence of columns here allows extracting the full data without having any tracking confusion problem. These experiments were performed on a cloudy day.

The performance metrics used were the same as in Experiment 2.

Figure 4.14 shows that the FPR were reduced, in some cases becoming 0, and in cases with prolonged time of life, the rate continued to be tiny compared with the original Pedestrian Detection Algorithm value.

The data initially had 5950 image files, because the XB3 camera (figure 2.2b) records at a rate of 16 fps (frames per second) and to avoid catching the pedestrian always on the same spot, the 5950 images were reduced to 1190, containing 901 ground truth positive bounding boxes.

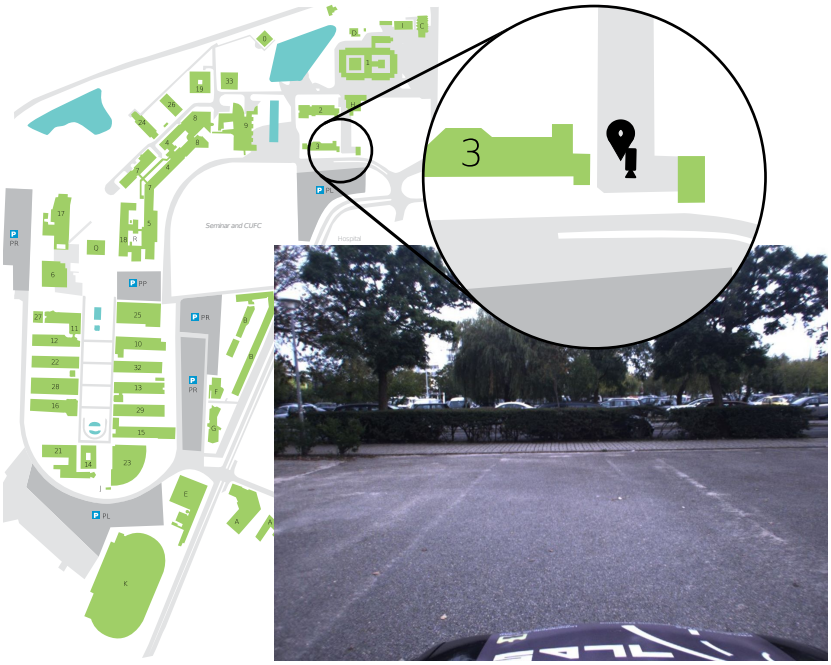


Figure 4.13: AtlasCar recording location, on the UA Campus, 3 - CESAM/CICECO/TEMA Department [UA]

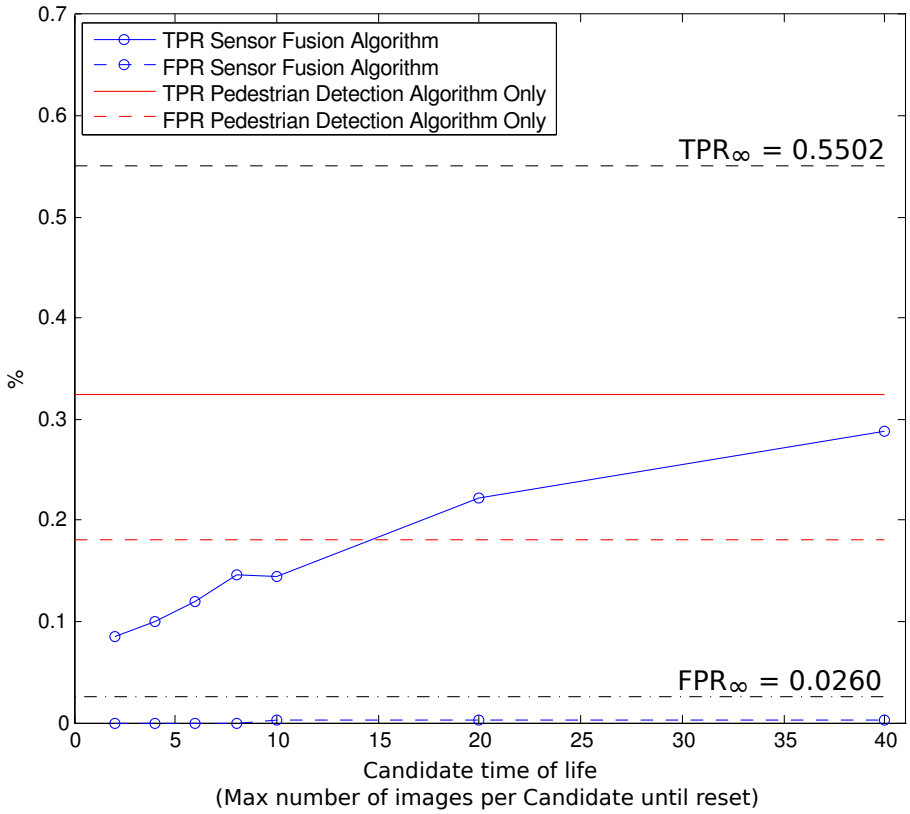


Figure 4.14: Experiment 3, TPR (True Positive Rate) and FPR (False Positive Rate) Results

Table 4.4: Experiment 3 detailed results

Life time	Sensor Fusion Algorithm		Pedestrian Detection Algorithm	
	TPR	**FPR**	**TPR**	**FPR**
2	0.0843	0	0.3242	0.1802
4	0.0997	0		
6	0.1204	0		
8	0.1465	0		
10	0.1447	0.0022		
20	0.2224	0.0022		
40	0.2884	0.0022		
∞	0.5502	0.0260		

Even though the research results have emerged as satisfactory, the tested time of life of "10" instead of growing as expected, decreased a little.

The pedestrian recorded data was similar to the one on a road, in spite of the fact that this data had been recorded in one steady place, where people passed not only in front but also on both sides of the car (see figure 4.15). The *Atlascar* is located at the (0,0) point. Each color tracked, corresponds to a potential pedestrian.

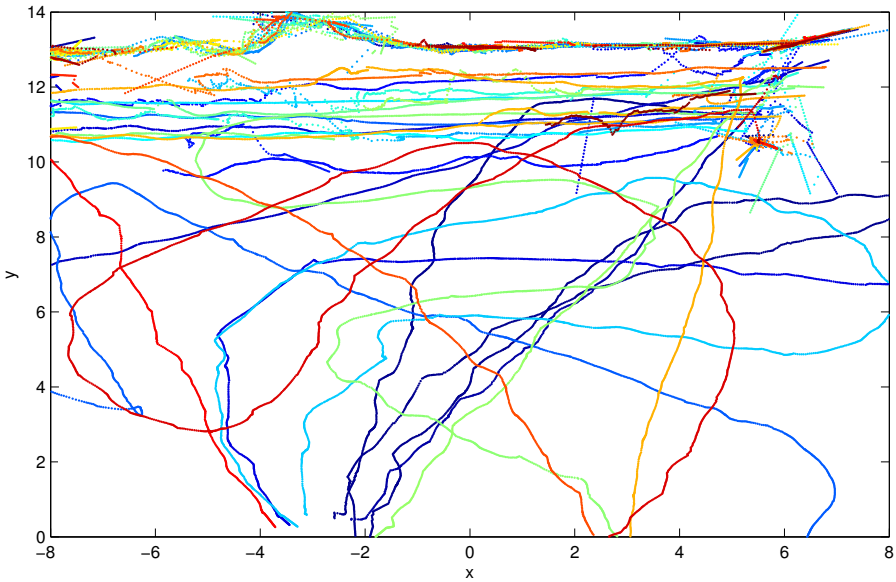


Figure 4.15: Pedestrians Route in *AtlasCar* dataset

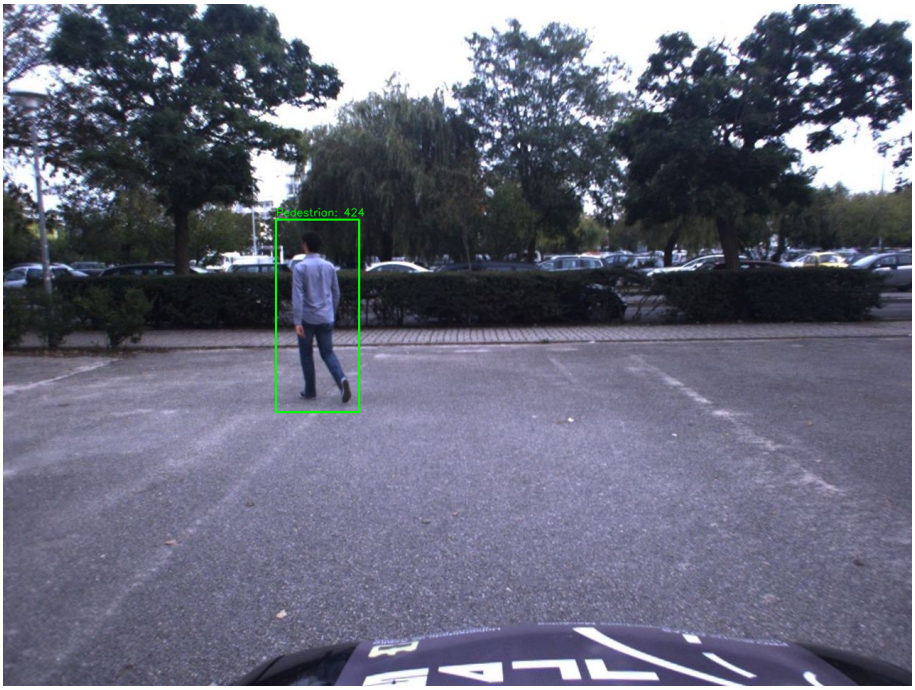


Figure 4.16: TP Fusion Algorithm Example 1

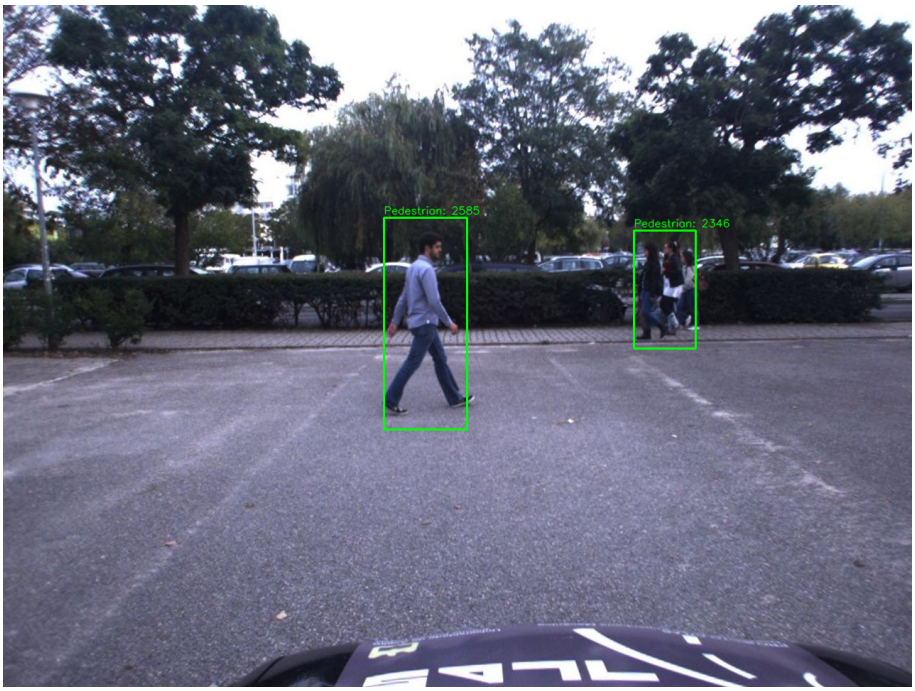


Figure 4.17: TP Fusion Algorithm Example 2

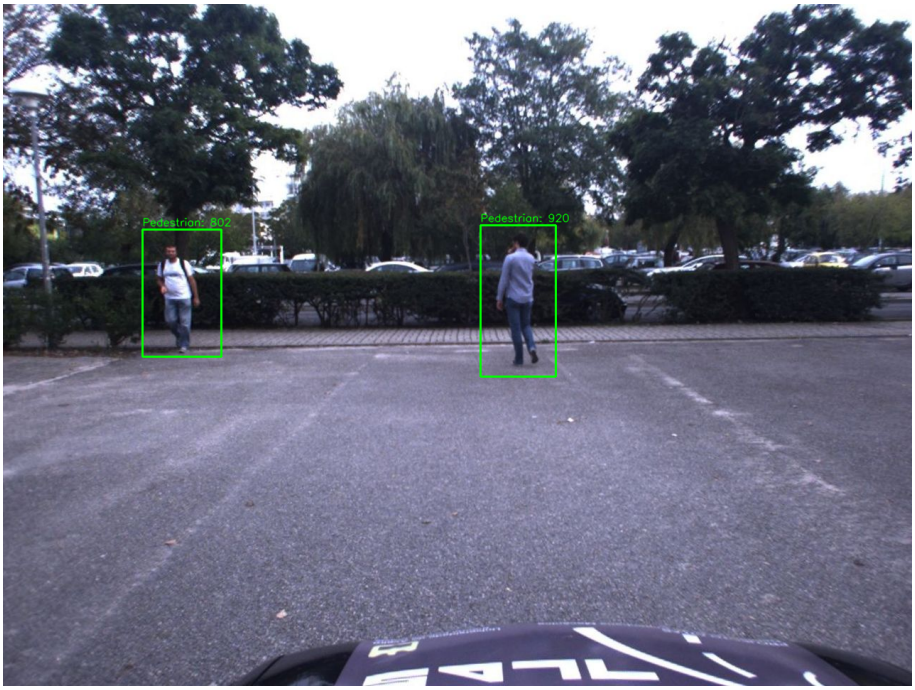


Figure 4.18: TP Fusion Algorithm Example 3

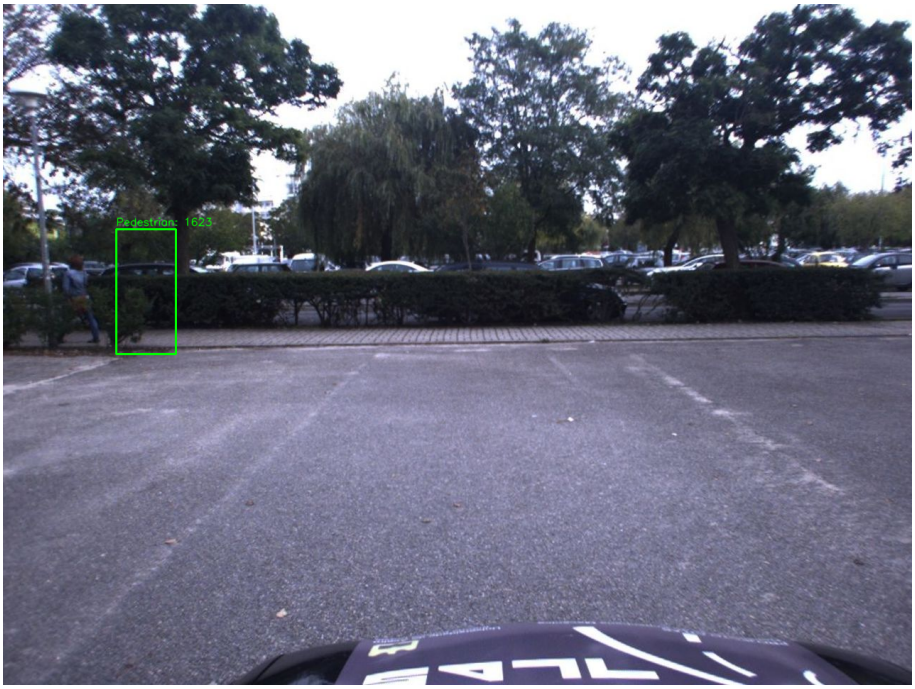


Figure 4.19: FP Fusion Algorithm Example - After a pedestrian passed the bushes, the classification continued to be shown for a short period of time

Figure 4.20: FP Pedestrian Detection Algorithm Example

The data also allowed a process time comparison, see table 4.5. The process time tends to decrease, when candidate time limit increases.

Table 4.5: Mean time (in seconds) to process an image

Life time	Sensor Fusion Algorithm	Pedestrian Detection Algorithm
2	1.1554	158.1968
4	0.8164	
6	0.8170	
8	0.7835	
10	0.5739	
20	0.5930	
40	0.5293	
∞	0.1466	

4.4 Discussion

The discussion on Experiment 2 lies on the quantification of the results. Although the research has allowed getting an average satisfactory dataset achievement in terms of quantification of the results, it proved to be a very small one (only with 220 images). Due to this, errors were induced (when changing the time of life) into the results by a slight percentage, causing a rapid oscillation (a rapid rise and descend).

Facts prove that the location was not very helpful because of the columns; as a matter of fact, the sensor fusion algorithm kept tracking some of the objects, even when crossing each

other. Due to this, some objects were classified wrongly. For example, pedestrians who walked in front of it, passed their classification on to columns.

The available evidence seems to suggest that even if this dataset was small, the results were very satisfactory, it means, both the FPR was reduced significantly compared to the initial Pedestrian Detection Algorithm and some of the experiments exceeded the Pedestrian Detection TPR. It seems fair to emphasize that it was unexpected; it proved to be very good, however.

The best performance accomplished with this dataset, increased the TPR by 6% and decreased the FPR by 8%.

The image processing time was decreased by a means of $\simeq$281%, comparing to the initial Pedestrian Detection algorithm.

FP detections decreased due to the LIDAR object selection.

In experiment 3, the algorithm was tested in a different environment and in different weather conditions.

Actually, this dataset was larger than the previous one (1190 images), which implies that the values vary in a more consistent way, instead of changing abruptly.

The best performance accomplished with this dataset, increased the TPR by 23% and decreased the FPR by 16%.

The image processing time was decreased by a means of $\simeq$633%, comparing to the initial Pedestrian Detection algorithm.

The results were very satisfactory, having a growing trend TPR and at the same time the FPR maintained low values.

Chapter 5

Conclusions and Future Work

Pedestrians are a very important issue in security driving. Due to the complexity of pedestrian detection, the usage of visual images is probably the best way to acquire its detection. However, it is still not very effective specially regarding CPU processing costs. With the support of a sensor (LIDAR), which helps to select potential pedestrians, visual search can be improved and faster at detecting pedestrians.

This project, where this work is inserted, combines a pedestrian tracking algorithm (LIDAR Processing) and a pedestrian detection algorithm (Image Processing) enhancing the best of the two sensors on pedestrian detection field.

Three different experiments were carried out to verify its validation. These experiments were performed in three different places with two different types of situations. First, on LAR (Automation and Robotics Laboratory) and then on the Department of Mechanical Engineering at the University of Aveiro facing the main entrance and finally onboard of *AtlasCar* at one of the main University pedestrian exits. On the first two situations a camera was vertically placed on top of the LIDAR.

This work features multi-thread process, which enables multi parallel potential pedestrian detection, while the program continues to run freely. It also stores each potential pedestrian properties and enables their constant moves (location update). Another feature of this work is the creation and elimination of candidates, potential pedestrians management which increases the overall algorithm performance because the eliminated candidates frees memory. Last but not the least, it helps the user to visualize potential candidates (Red bounding box) and actual classified pedestrians (Green bounding box) to an output image.

The first experiments relies on testing and improving some visual aspects as well as the tuning of some parameters to decrease the CPU processing cost. Here, a candidate "life time" parameter was also created that resets the candidate vector every time the image counter reaches the given life time limit. For example, if the life time is "10", candidate classification is reset in every ten received images. This parameter was also implemented on the second and third experiment. During the second experiment, the best performance was achieved while having a candidate time of life of "20" which increased the detection TPR (True Positive Rate) by 6% and decreasing the detection FPR (False Positive Rate) by 8%. On the third experiment, the best performance was obtained having no candidate time of life resulting in an increase of the detection TPR by 23% and a decrease of the detection FPR by 16%.

The second and third experiments were carried out outdoors in order to make it more realistic and reliable.

It was observed that pedestrian detections are influenced by environment light, where the second experiment had a better performance compared to the third one. Shadow areas seem to be responsible for most missed pedestrians.

Sensor fusion method showed that the CPU memory (RAM) cost can be decreased (2GB down from 12GB, in a 1280x960 image) as a result of images that do not need to be full sized, but instead, potential pedestrian size image. As result, the CPU processing cost was substentially decreased allowing faster detections and a decreasing regarding the false positive rate because the searching areas can be narrowed to the important ones with the LIDAR help.

The main goals of this work were achieved by: a re-parametrization of a Pedestrian Detection Algorithm becoming a Single Pedestrian Detection Algorithm allowing pedestrian detection on images of any size; A development of a manual calibration tool between the LIDAR and the camera; A semi-automatic calibration tool was built and tested, however, unsuccessful; Creation of an application that combines LIDAR and vision data in order to actively detect pedestrians by means of image localized search.

Achieving all main goals means that the sensor fusion of LASER and Image processing application for active pedestrian detection was successfully implemented.

The greatest contribution of this project was that detection processing time decreased substantially. The incorporation of the system in the *AtlasCar* is a feature that had not yet been developed since the begining of this work. This project can, and most certainly will, be improved in the future helping not only pedestrians but also car detections in real time situations aboard *AtlasCar*.

Experiments were conducted in a Asus N53SV notebook, for more informations on the Hardware and Operating System see appendix B.

The present document was written in English to extend language knowledge.

Future Work

In the future, the fusion situation may be able to:

Predict spaces where the laser cannot reach and entrust the camera to test if anyone near exists, for example, when two cars are parked parallel to the sidewalk. The gap between the two cars (critical point) is often a path chosen by the pedestrian to cross the road and surprise the driver, as illustrated in Figure 1.11.

Take advantage of pedestrian classifications to enlarge detection datasets for further usage;

Classify different types of object, modifying objects parameters and changing detection datasets;

Use the expected time of collision to dynamically studying the pedestrians and choose whether or not to classify. This matter can be use to predict if a pedestrian is walking towards or moving away from the car. For example, if a pedestrian is in an eminent danger of colliding with the car, that pedestrian needs to be the top priority for classification and so on.

References

[Almeida, 2010] Almeida, J. (2010). Targe tracking using laser range finder with occlusion. Master's thesis, University of Aveiro.

[Benenson et al., 2012] Benenson, R., Mathias, M., Timofte, R., and Van Gool, L. (2012). Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910. IEEE.

[Blackman and Popoli, 1999] Blackman, S. S. and Popoli, R. (1999). *Design and analysis of modern tracking systems*. Artech House radar library. Artech House, Boston, London.

[Broggi et al., 2009] Broggi, a., Cerri, P., Ghidoni, S., and Grisleri, P. (2009). A New Approach to Urban Pedestrian Detection for Automatic Braking. *IEEE Transactions on Intelligent Transportation Systems*, 10(4). Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290131`.

[Bumblebee] Bumblebee. Point Grey Bumblebee® XB3. Visualized in 21-05-14. Available from: `http://ww2.ptgrey.com/stereo-vision/bumblebee-xb3`.

[Caltech] Caltech. Camera Calibration Toolbox for Matlab. Visualized in 15-09-14. Available from: `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[Dollar et al., 2010] Dollar, P., Belongie, S., and Perona, P. (2010). The fastest pedestrian detector in the west. In *Proceedings of the British Machine Vision Conference*. BMVA Press. Available from: `http://vision.ucsd.edu/~pdollar/files/papers/DollarBMVC10FPDW.pdf`.

[Douillard et al., 2007] Douillard, B., Fox, D., and Ramos, F. (2007). A spatio-temporal probabilistic model for multi-sensor object recognition. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4399537`.

[Everingham et al., 2009] Everingham, M., Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2009). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2). Available from: `http://link.springer.com/10.1007/s11263-009-0275-4`.

[Gandhi and Trivedi, 2007] Gandhi, T. and Trivedi, M. M. (2007). Pedestrian protection systems: Issues, survey, and challenges. *Intelligent Transportation Systems, IEEE Transactions on*, 8(3):413–430.

[Gidel et al., 2009] Gidel, S., Blanc, C., Chateau, T., Checchin, P., and Trassoudaine, L. (2009). Non-parametric Laser and Video Data Fusion: Application to Pedestrian Detection in Urban Environment.

[Guan et al., 2009] Guan, C. H., Gong, J. W., Chen, Y. D., and Chen, H. Y. (2009). An application of data fusion combining laser scanner and vision in real-time driving environment recognition system. *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, 6.

[Li et al., nd] Li, H., Yang, M., and Qian, H. (n.d). Camera and laser scanner co-detection of pedestrians. Visualized in 05-02-2014. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.186.7684&rep=rep1&type=pdf`.

[Logitech] Logitech. Logitech - HD Webcam C310. Visualized in 21-05-14. Available from: `http://www.logitech.com/en-us/product/hd-webcam-c310`.

[Ludwig et al., 2011] Ludwig, O., Premebida, C., Nunes, U., and Araujo, R. (2011). Evaluation of boosting-SVM and SRM-SVM cascade classifiers in laser and vision-based pedestrian detection. *In Intelligent Transportation Systems, ITSC. IEEE International Conference.* Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6082909http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6082909`.

[Parzen, 1962] Parzen, E. (1962). Parzen Window Density Estimation. Visualized in 28/05/14. Available from: `http://bayes.wustl.edu/Manual/parzen62.pdf`.

[Premebida, 2012] Premebida, C. (2012). *Pedestrian Detection Using Laser and Vision.* PhD thesis, University of Coimbra. Available from: `http://webmail.isr.uc.pt/~cpremebida/files_cp/PhdThesis.pdf`.

[Premebida et al., 2009] Premebida, C., Ludwig, O., and Nunes, U. (2009). Exploiring LIDAR-based features on pedestrian detection in urban scenarios. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC.*

[Sick] Sick. Sick LMS 151. Visualized in 21-05-14. Available from: `https://www.mysick.com/eCat.aspx?go=DataSheet&Cat=Row&At=Fa&Cult=English&ProductID=35835`.

[Silva, 2013] Silva, P. (2013). Visual Pedestrian Detection using Integral Channels for ADAS. Master's thesis, University of Aveiro.

[UA] UA. Campus UA Map. Visualized in 08-10-14. Available from: `http://www.ua.pt/campusdaua`.

[UniversityofRegina] UniversityofRegina. Confusion Matrix. Visualized in 05-10-14. Available from: `http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html`.

[Zhao et al., 2009] Zhao, H., Zhang, Q., Chiba, M., Shibasaki, R., Cui, J., and Zha, H. (2009). Moving Object Classification using Horizontal Laser Scan Data. *In proceedings of the 2009 IEEE international conference on Robotics and Automation, ICRA'09.*

Appendix A

Usage Instructions

Launch files were created to launch the necessary ROS nodes for the program to run.

A.1 Sensor Fusion Program

To enable the full usage of the Sensor Fusion Program, two roslaunch programs need to be started. They are:

- generate_single_laser_planar_cloud_with_tf.launch

This launch file transforms the SICK laser frame in a *AtlasCar* front bumper frames, because the two front lasers are combined in the *AtlasCar* front bumper frame. Subscribes "/snr/las/2/scan" and publishes the "/scan_cloud" message

It also starts the MTT program that subscribes the transformed laser points into MTT targets ("/scan_cloud"), publishing them with the "/new_targets" name. The next code shows its parameters.

```
<launch>
<!-- Inside contains a comment -->
<!--    Remap AtlasCar transformations (/tf to /trf/frames) needs input:
    "/trf/frames" -->
    <remap from="/tf" to="/trf/frames"/>

    <group ns="/pcp/fus/planar_pc">
<!--   Left (/snr/las/2/) or Right (/snr/las/3/) Laser; input needed:
    "/snr/las/2/scan" -->
        <remap from="/laserscan0" to="/snr/las/2/scan"/>

<!--        Output tracking frame -->
        <remap from="/tracking_frame" to="/atc/vehicle/center_bumper"/>
        <remap from="/pc_out" to="/scan_cloud"/>

<!--    Starts the laser transformation to output tracking frame (AtlasCar) -->
        <node name="planar_pc" pkg="mtt" type="simple_planar_pc_generator_atlasmv"
            output="screen">
            <param name="output_frequency" value="200.0"/>
```

```
            <param name="perpendicular_treshold" value="0.15"/>
            <!-- simple_planar node output: "/scan_cloud"-->
        </node>

        <!-- This starts multi target tracking for stage laser measurements-->
        <node name="node" pkg="mtt" type="mtt_pedect">
            <remap from="/points" to="/scan_cloud"/>
            <remap from="/markers" to="/ids"/>
            <!-- MTT node output: "/new_targets"-->

        </node>
    </group>
</launch>
```

- sensor_fusion.launch

This launch file launches an image republisher, which rectifies the camera images. It subscribes the "/xb3/right/image_raw" and the "/xb3/right/camera_info", publishing the already rectified "/xb3/right/new_info/image_rect_color" and the "/xb3/right/new_info/camera_info".

It also starts the sensor fusion node where the classifier location is used for the Single Pedestrian Detection Algorithm.

The sensor fusion node subscribes the "/xb3/right/new_info/image_rect_color", the "/xb3/right/new_info/camera_info" and the "/new_targets" (provided by the MTT node). The next code shows its parameters.

```
<launch>

<!--  XB3 proc image republisher -->
  <!--  XB3 proc image republisher input: "/xb3/right/image_raw/compressed " -->
    <node ns="/xb3/right" name="republish_right" type="republish"
        pkg="image_transport" args="compressed in:=image_raw
        out:=image_decompressed"/>

    <!-- Camera info republisher (replace camera info with calibrated camera info)
        -->
    <node ns="/xb3/right" name="info_changer_right" type="camera_info_republisher"
        pkg="multisensor_aquisition" args="image_raw:=image_decompressed">
        <param name="camera_calibration_file"
            value="package://xb3/calibrations/right_full_resolution.yaml"/>
        <param name="camera_name" value="right"/>
    </node>

    <node ns="/xb3/right/new_info" name="right_camera_rectification"
        type="image_proc" pkg="image_proc"/>
  <!--  XB3 proc image republisher output: "/xb3/right/new_info/image_rect_color"
      and "/xb3/right/new_info/camera_info" -->

<!-- Sensor Fusion Program -->
```

```
  <!--  Sensor Fusion Program input: "/xb3/right/new_info/image_rect_color",
      "/xb3/right/new_info/camera_info" and "/new_targets". If one wants to observe
      the laser projected points in the image, the "/scan_cloud" subscriber topic
      needs to be uncomment on sensor_fusion.cpp -->
    <node name="sensor_fusion_prog" pkg="multimodal_pedestrian_detection"
        type="sensor_fusion" output="screen">

<!--   Classifier Location  -->
        <param name="classifier"
            value="/.../trained_boost_15Kf_2000w_19Ks_m8_M64_boot3.xml"/>

    </node>
</launch>
```

How to run the sensor fusion program

The sensor fusion program can be executed by playing a recorded rosbag, when it contains every message topic mentioned above (commented code), or simply run it aboard the *AtlasCar*, applying the same conditions.

Provided access to the LAR tool kit, follow the next steps to run the program:

1. Open a terminal on Ubuntu.
2. Go to LAR tool kit primary folder (ex. "lar4", where is only seen the "src" folder) and type "catkin_make" to compile every single package, creating a "build" and "devel" folder. To compile each package separately one can do:
   - If one wants to compile only the "MTT" package, inside the "build" folder enter "cd /perception/planarobstacles/mtt/" and type "make".
   - To compile only the "sensor fusion" package, in the "build" folder enter "cd /perception/pedestrians/multimodal_pedestrian_detect" and type "make".
3. Run the program by entering "roslaunch multimodal_pedestrian_detection sensor_fusion.launch".
4. On another terminal enter "roslaunch multimodal_pedestrian_detection generate_single_laser_planar_cloud_with_tf.launch".

A.2 Calibration Process

For calibration purposes, a program was created to calculate and output the transformation matrix (${}^{C}[R|t]_{FB}$), when given two .txt files with image and object points coordinates (input).

If one wants to execute the calibration launch file, the "multimodal_pedestrian_detect" package needs to be compiled (process mention in Section A.1). The next code shows the "calibrate_extrinsic_parameters.launch" file and its parameters.

```
<launch>

    <node name="Client_extrinsic_calibration" pkg="multimodal_pedestrian_detection"
        type="calibrate_camera_extrinsic" output="screen">

<!--  Image Points Location, file = .txt (Input) -->
      <param name="imagePoints1FileName"
          value="/.../imagePointsAuto_1280_960_xb3.txt"/>

<!--  Object Points Location, file = .txt (Input) -->
      <param name="objectPoints1FileName"
          value="/.../objectPointsAuto_1280_960_xb3.txt"/>

<!--  File storage Location, file = .yaml (Output) -->
      <param name="fileStorageYAML"
          value="/.../bumblebeeXB3_1280_960_transform_matrix_to_2Dlaser.yaml"/>

    </node>
</launch>
```

To execute this launch open a terminal in Ubuntu and type "roslaunch multimodal_pedestrian_detection calibrate_extrinsic_parameters.launch".

Appendix B

Hardware and System Notes

This work was conducted on a Asus N53SV notebook with the following specifications:

- Hardware
  - Intel® Core™ i7-2630QM CPU @ 2.00GHz × 8
  - 6GB (3+3) DDR3 1333 MHz SDRAM
  - NVIDIA® GeForce® GT 540M with 1GB DDR3 VRAM
  - SSD Samsung 840 EVO (250GB) hard drive
- System
  - Ubuntu Linux version 12.04 LTS (64 bits)
  - Linux kernel version 3.11.0-26-generic
  - ROS Hydro version