Master of Science in Aerospace Engineering
Major: Aircraft Systems and Control

Vision based object detection and avoidance

Written by: Andrés GONZÁLEZ MORENO, Álvaro LÓPEZ GUTIÉRREZ
Supervised by: Regine LECONTE (ISAE-SUPAERO), Jean-Baptiste CHAUDRON (ISAE-SUPAERO), Vincent LECUBRIER (Sterblue)

Abstract: The present project aims to research into the potential development of Unmanned Aerial Vehicles (UAV) performance and autonomy. Particularly, the project focuses on an algorithm for obstacle detection and avoidance in real time, for which the ways of data obtaining will be a stereo camera, able to get the depth of an image.
High performance will be the central objective all along the development of the project because the computations and autonomous actions are thought to be performed in real-time. In order to achieve the specified targets, the embedded device in charge of all the data processing will be a General Purpose Graphic Processing Unit. Thanks to its capability to carry out parallel computations, the efficiency of the whole system can be really high. That is why the algorithm developed must be thought to be computed in parallel. To do so, specific parallel computation libraries will be used.
This project gathers the steps followed to obtain the final results. Firstly, the motivation and objective of the project are presented. Secondly, there is a presentation of the resources (hardware and software) used. Then, the state-of-art has been researched in order to gather information about the current situation in the fields of study. Plus, a deeper study into the performance of GPGPU was carried out. Moreover, several algorithms have been developed to create an autonomous obstacle avoidance system. This software is thought to be performed in a ROS environment so that multiple data and devices can be interconnected. Finally, some tests (simulation and experimental) have been performed obtaining high-efficiency results. With a performance of approximately 60 Hertz, the system is able to avoid obstacles with smooth trajectories until reaching the target position

Keywords: Avoidance, GPGPU, obstacle detection, parallel computation, real-time, stereo camera, UAV.

Contents

1 Introduction

1.1 Motivation of the project

Since the very beginning when automatic aerial devices and vehicles were created, their applications have unavoidably spread to almost any field of use. That is why, the Unmanned Automatic Systems, normally abbreviated as UAS, increasingly require more and more sensors in order to improve their implementation and performance. Thanks to the aforementioned development, the performance of these devices is nowadays object of study as the final goal is always to make this vehicles completely automatic and reliable.

Laying aside the technical development and focusing on the economical one, many enterprises have taken advantage of the current situation in the automatic vehicles field and have started several ambitious projects. One of these enterprises is Sterblue, a French start-up, which aims to carry out high performance inspections in power lines and wind turbines, not only onshore but also offshore. In order to achieve this goal, the systems used will be fully autonomous. Therefore, not only will the quality but also the efficiency of the services be noticeably improved. For example, the current price of an inspection offshore would be around 800 € and with this type of system only two or three inspections per day would be possible. In contrast, the objective of Sterblue is to fully develop an entire innovative automatic aerial system of inspection, so that the price could be reduced to 400 € and the number of inspections per day could be increased up to 16.

One basic requirement for the fully autonomous operation of a UAV is the obstacle detection and avoidance, as the integrity of the system and entities surrounding the vehicle has to be guaranteed, which will be the main topic covered in this project.

The most remarkable technologies being used for the detection of the obstacles are ultrasound sensors, LIDAR and stereo cameras. Ultrasound sensors have interesting performances in regard to their accuracy, but they are limited in range. LIDAR is a more powerful technology, having an impressive range of use and high reliability. However,i has some drawbacks such as the cost, weight and, depending on their type and application, moving parts. Both of these technologies have also a dimensional limitation on the data acquisition, being complicated to obtain a 3 dimensional consistent mapping of the space around them. Therefore the interest lies on the last technology mentioned, the stereo vision. Stereo cameras are able to map in three dimensions consistent volumes of space, and the most recent models are lightweight (important for their application in drones) and relatively cheap. Their drawbacks are the limited operation conditions with respect to the illumination of their surroundings and the high computational cost of the stereo vision algorithms.

However, when taking into account another technology being recently developed into more affordable prices and more lightweight designs, the drawbacks of the stereo cameras are

reduced. This technology are the General Processing Graphic Processing Units, GPGPU. Their specific parallel computing architecture is highly compatible with computer vision and artificial intelligence algorithms, and it is able to speed up the stereo vision algorithms up to real time computation.

For this reasons, the combination of these two previous technologies, the stereo vision and the GPGPUs, opens a fairly recent field of study that has great potential for the development of a real time, on board lightweight obstacle detection and avoidance system.

1.2 Objectives

The project will focus, as required by Sterblue, on the development of an integrated system which is able to perform the obstacle detection and avoidance, on board of the vehicle and in real time.

To achieve this goal, the project's pillar will be the study of a visual detection system based on the use of stereo cameras in order to detect the surrounding environment of the vehicle. With base on the data stream received, an algorithm will be developed so that the vehicle is able to compute an avoidance trajectory and act automatically in consequence. The architecture used for the design of the algorithms will be a GPGPU.

The secondary objectives of this project, according to the needs of Sterblue and mentioned in priority order are:

- To come up with an appropriate system and to develop the corresponding algorithms to carry out an automatic precision landing with an UAV.

- To implement the algorithms which let the automatic vehicles follow specific patrons so that the main trajectory to follow is based on these particular patrons. To do so, of course we should take into account the previous steps and also implement the obstacle avoidance objectives.

- To be able to create algorithms of object recognition in order to identify specific complex objects and shapes.

1.3 Structure of the report

This report aims to explain the basics of the project. The resources (hardware) that were assigned to this project will be presented. A study about the state-of-art is carried out to well understand the background in the fields of study. As part of the initial experimental work some basic testing is done to show the comparison between the performances of CPU and GPGPU in highly parallelizable algorithms. After, the complete algorithm dedicated to the obstacle avoidance is explained. Finally, both theoretical and experimental testing are carried out with the algorithm, and the results are presented and analyzed. The report

finishes with the conclusions made after analyzing the results in addition to a comment for future work that might be conducted on the topic.

2 Resources

In this section, we will explain the main available resources used in the project.

2.1 Stereo cameras

As it was already explained the main sensors of the systems are stereoscopic cameras. The models tested are:

- *Intel RealSense R200* from Intel [1]: Stereoscopic camera whose main feature is the computation of the depth by means of infrared sensors. The camera showed high performance thanks to its embedded ASIC. However, the short range of this device (0.5m - 3.5m) represents a strong limitation to its potential uses in obstacle avoidance. The camera can be observed in figure 1.

Figure 1: Camera Intel RealSense R200.

- *Zed Stereo Camera* from Stereo Labs [2]: Likewise the previous camera, this device allows the user to detect the depth and the distance in the images captured. However, the main difference with respect to the Intel RealSense is that the camera is designed to be highly compatible with the GPGPU architecture, which improves the performance in data processing. Its range is significantly superior to the R200's one: 0.7m to 20m. The camera can be observed in figure 2.

Figure 2: ZED stereo camera by Stereolabs.

These differences in terms of features and the fact that the house developing the ZED camera, StereoLabs, is focusing its research in the development of GPGPU based computer

vision algorithms with powerful applications for autonomous navigation, have been the main reasons that motivated the decision of using the ZED stereo camera as the most suitable device for our application. The camera can be observed in figure 2.

References [2] and [1] can provide more information about these models.

2.2 GP-GPU

Another key element in the present project is the General Purpose Graphic Processing Unit. It will be demonstrated that the performance of a GPU is needed due to the large amount of data that the devices will have to handle. The GPGPUs being currently developed with the best performance/(weight*cost) ratio are the Nvidia Jetson series, specifically designed for computer vision and AI (Artificial Intelligence) in autonomous machines and vehicles.

Several models have been released by Nvidia, among which the 3 latest have been tested for this project. These models are the Jetson TK1, Jetson TX1 and the most recent and used for the final version of the project, the Jetson TX2. A picture of this module is shown in figure 3.

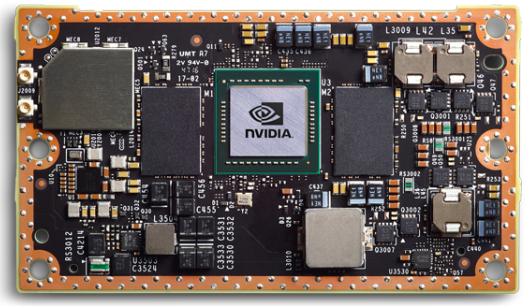

Figure 3: Jetson TX2 Module from NVIDIA.

The specifications of the modules of the most recent series, the TX, are shown in table 1.

	Jetson TX2	Jetson TX1
GPU	NVIDIA Pascal™, 256 cœurs CUDA	NVIDIA Maxwell ™, 256 cœurs CUDA
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2	Quad ARM® A57/2 MB L2
Vidéo	Encodage 4K x 2K 60Hz (HEVC) Décodage 4K x 2K 60Hz (support 12-Bits)	Encodage 4K x 2K 30Hz (HEVC) Décodage 4K x 2K 60Hz (support 10-Bits)
Mémoire	8 GB 128 bit LPDDR4 59.7 GB/s	4 GB 64 bit LPDDR4 25.6 GB/s
Affichage	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4	2x DSI, 1x eDP 1.4 / DP 1.2 / HDMI
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.1 (1.5 Gbps/Lane)
PCIE	Gen 2 | 1x4 + 1x1 OR 2x1 + 1x2	Gen 2 | 1x4 + 1x1
Stockage	32 GB eMMC, SDIO, SATA	16 GB eMMC, SDIO, SATA
Autres	CAN, UART, SPI, I2C, I2S, GPIOs	UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0	
Connectivité	1 Gigabit Ethernet, Connexion aux appareils compatibles 802.11ac WLAN et Bluetootha	
Caractéristiques Mécaniques	50 mm x 87 mm (Connecteur carte à carte compatible 400-Pin)	

Table 1: Specifications of the TX series modules.

2.3 Robotic vehicles

A surface robot will be used to test the system. The main motivation for this choice is safety during the testing procedure, as the test obstacle avoidance will require to drive the vehicle into obstacles deliberately. A malfunction of the system (possible during testing) would result catastrophic for an aerial vehicle, and not be a problem for a surface robot with protections and a reasonable speed.

The vehicle used for these tests is a iRobot Create, whose characteristics are shown in table 2. An image of the top view of the iRobot Create is shown in figure 4.

iRobot Create specifications	
Weight [kg]	2,9
Diameter [m]	0,343
Linear speed saturation [m/s]	[-0.3,0.3]
Angular speed saturation [rads/s]	[-2.765,2.765]

Table 2: Summary of the main characteristics of the iRobot Create model used in the experiments.

Figure 4: Top view of the iRobot Create.

2.4 Others

The Optitrack [3] is a system conceived to track and give accurate positioning of objects within a space. The objects are identified thanks to a collection of markers arranged in an specific display. The system is composed by a set of cameras which track the balls and the position is estimated thanks to the triangulation of the positions tracked by each camera. Then, as long as these balls are carried by the vehicle, its position will be accurately known.

Optitrack hardware is composed by a set of high speed cameras distributed around the working space. From different points of view and thanks to a real-time body solver it is able to deliver 6 degrees of freedom positioning information over wide spaces.

3 State of the art

Several research projects have been already conducted in the field of image processing applied to the automation of unmanned vehicles. More precisely, the use of stereo vision algorithms has already been proven to be able to compute the depth map of a given pair of images. The precision is enough for the detection of common sized objects that may represent a threat for the integrity of the vehicle. This method has been widely used in investigations conducted for both UAVs and cars, as the need for automation in this two fields is increasing every year. An example of this is the zFAS TTTech project conducted at AUDI [4], which aims to integrate all the information obtained by the car sensors with the guidance system of the vehicle. Moreover, all of it will be embedded into a single electronic board, based in parallel processing of the data.

In the following sections, the research done in the fields of study concerning our project will be presented.

3.1 Stereo vision

Stereo vision consists on obtaining useful information from the comparison between two images of the same scene. In this, and the more common, case, this extra information will be the depth of the pixels. The pillars of the depth maps algorithms are the computation of the disparity between pixels [5] and the triangulation of the pixels into a three-dimensional space [6].

The disparity consists on the comparison between several images, generally two, from several cameras or lenses. Basically, after taking the two images, the algorithm is configured to pick a certain area of one of the images and compare it with the other image by looking for the least different region of pixels. In the algorithm is also specified the zone of searching because knowing that both images are the same there is no point in comparing all blocks of one image with all blocks of the other, but with a close region.

The numerical comparison is performed through the SAD (i.e. Sum of Absolute Differences) method. The first step of this method is to convert the image in a unique scale-of-values matrix, thus a black and white image. Afterwards, the values of the selected region in the first image are subtracted from the values of all the regions of the second image (with which we are doing the comparison). Finally, in every result matrix, all the absolute values are added up so that we get a single value per comparison. Then, it can be deduced that the lowest value belongs to the comparison of two equal pixel regions. Of course in the stereo cameras, this base algorithm is slightly modified in order to overcome possible drawbacks such as noise or excessive light exposure.

With this explanation, the only aspect missing now is how to get the depth. For a point in the space, we will have two images of it, one per camera. Because the distance between

the projectors is known and we know exactly where that point is in both images captured (that is, the disparity) it is possible to know the third coordinate, the depth, thanks to a process of triangulation. A graphical representation of this procedure can be observed in figure 5.

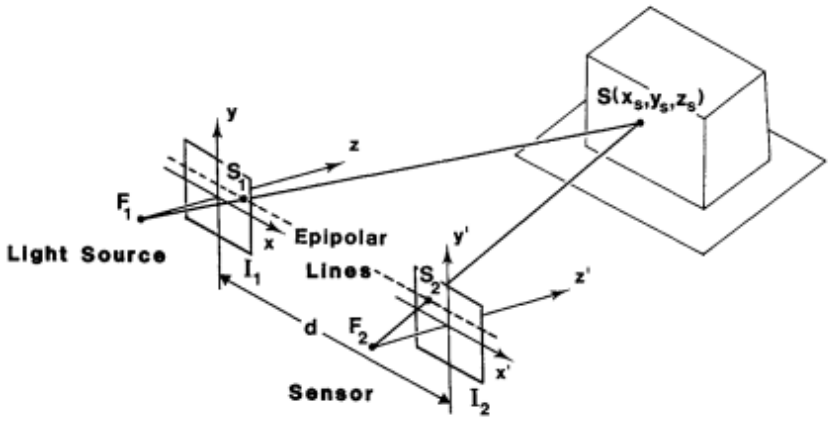

Figure 5: Scheme of triangulation process with two focal points.

The depth maps are a key element for this project. By knowing the distance of the points around our device, it could be possible not only to develop and perform the optimum trajectories (and to avoid the obstacles) but also to create complete 3D models whose applications could be extended even out of the scope of the present project.

The main challenges in the stereo depth map reconstruction are related to the trade-off present between the quality of the depth map and the speed of the algorithm.

The quality of the depth map is mainly based on the uncertainty derived from the disparity computed between the pixels of both images, which has to be carefully taken into consideration. Depending on the method used and the characteristics of the analyzed environment (brightness, repetitive features, occlusions..), the noise present in the depth map can reach unacceptable values for the reliability of the object avoidance system. A post filtering technique is often used to remove the number of outliers and to smooth the final image.

The speed of the algorithm can represent a highly restrictive constraint for the object avoidance systems in unmanned vehicles. The reason for this is obvious: as the vehicle operates in real environments, the system must be able to compute the potential object collisions with a frequency high enough for the speed at which the vehicle is moving. Due to the size of the elements that are being operated, i.e. large size pixel arrays, the iterative computations can turn into heavy computer processes.

There exists a large number of different methods used to compute the disparity between

two stereo images, which every year seems to be more focused on the deep learning field due to the high accuracy shown by the latest developments. However, when it comes to satisfying the image quality and speed trade-off, the block matching method (simpler but faster than the deep learning based ones), provides more than sufficient results in most of the cases.

For some extra information about depth and disparity maps computation, refer to [5] and [7].

Other algorithms and libraries

The aim of this subsection is to complete the previous ones with more available libraries and algorithms that are of interest for the computer vision field.

OpenCV [8] is a library that gather a useful collection of computer vision algorithms ready for use, along with tools that make the image processing easier. It supports the main programming languages (C, C++, Python). It can support real-time applications and also multicore processing. Particularly, and quite beneficial for our project, it has already implemented a CUDA compatible library that allows to use all the image processing algorithms with the parallel processing capabilities of CUDA.

3.2 Obstacle Avoidance and Motion Planning

In this section the different Obstacle Avoidance and Motion Planning algorithms researched will be named and their characteristics explained.

3.2.1 3D trajectory optimization

Once the objects have been properly detected, the trajectory that had been computed for the vehicle must be redefined in order to avoid the obstacle. For this purpose, the most extended technique consists on the three-dimensional discretization of the surroundings of the vehicle and the computation of an optimized trajectory that avoids collision with the detected objects.

Various trajectory computation algorithms have to be considered. From [9] , we extracted some surveys. The most remarkable algorithms that have been found are the Random Belief Trees (RRBT), the Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE), the Covariant Hamiltonian Optimization and Motion Planning (CHOMP) and the Stochastic Trajectory Optimization for Motion Planning (STOMP).

Again, the selected method must meet the required performances for real time applications and thus, the quality of the feature provided by the algorithm may be decreased in favor of its performance. it will be taken into account the potential parallelization of the algorithm, as that would make it compatible with the GPGPU architecture.

3.2.2 Push broom

A singular but interesting trajectory re-computation technique is the push broom algorithm. This method, used in [10], is based on the computation of a single window of the depth map, located at a sufficient distance from the drone to have the necessary time to perform the avoidance maneuver. As the obstacles are only detected at a specific distance and no 3D reconstruction of the surroundings is needed, the computational load of this method is significantly lower. Due to this property, this method may be significantly interesting for high speed real-time applications.

3.3 Parallel programming

It has already been mentioned the capabilities of the parallel architecture of the GPGPUs and their potential benefit for the project. Now, the parallel computation redefines part of the usual programming methods, as the multi-thread architecture avoids the repetition of operations which are independent from each other.

The library that makes this new programming method easier is called CUDA (Compute Unified Device Architecture) [11]. CUDA was created by Nvidia and contains all the functions needed to manage the memory allocation and transfers between CPU and GPU, define functions (kernels) on the GPU and many other sub-functions that simplify the process of programming on a parallel architecture.

Nvidia also provides the users with a lot of other libraries in order to be able to work with their devices. As an example, the library called CUBLAS [12], an adaptation of BLAS (Basic Linear Algebra Subprograms) to CUDA, speeds up even more every linear algebraic operation that may be computed in parallel.

3.4 Robotic interfaces

Regarding the interfacing of the different subsystems of our robotic vehicle, ROS (Robotic Operative System) [13] has been the selected software for our experiments. It provides a standardized communication protocol, robust and user-friendly, with a large set of tools and support from any software library involved with robotics.

4 GPU/CPU experimental comparison

In order to put of all of this into practise, we did several tests to check out how these tools work and which is the best way of using them. In the following graph, figure 6, we show the results obtained after computing the matrix product between n*n size matrices with both BLAS and CUBLAS. The main difference between these two libraries is that CUBLAS configures the computation in order to perform it with several parallel threads whereas BLAS only uses the CPU, even if this last has more than one core. We should remark that the data handled is of the type double floating point.

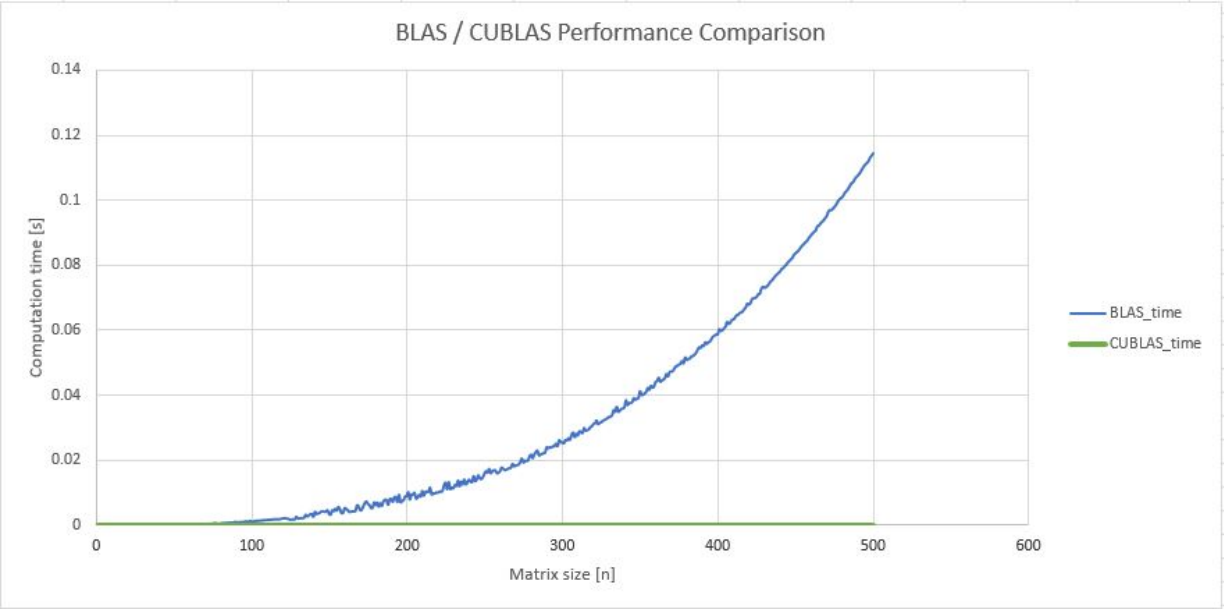

Figure 6: Comparison between BLAS and CUBLAS performances (performed in a Tesla C2050).

As it can be observed, for relatively small sized matrices, the performance is similar but there is a point (nxn = 25x25) in which the performance of CUBLAS shoots up and exceeds more than enough BLAS performance. In conclusion, when dealing with high quality images and videos, we will not be operating with 25*25 matrices but much larger ones. Therefore, GPGPUs offer a clear advantage that will allow the performance of the algorithm to be in frequencies acceptable for real time applications.

Once the comparison between CPU and GPU has been established, we compared the performances of different models of GPGPUs. The first test compares the non-embedded GPGPU Tesla C2050 with one from the Nvidia Jetson series, embedded on a system on chip and the one used in our project. The results from this comparison can be observed in figure 7.

The specifications of these two GPGPUs in the aforementioned examples are displayed in table 3, where the superiority of the non-embedded Tesla C2050 can be observed.

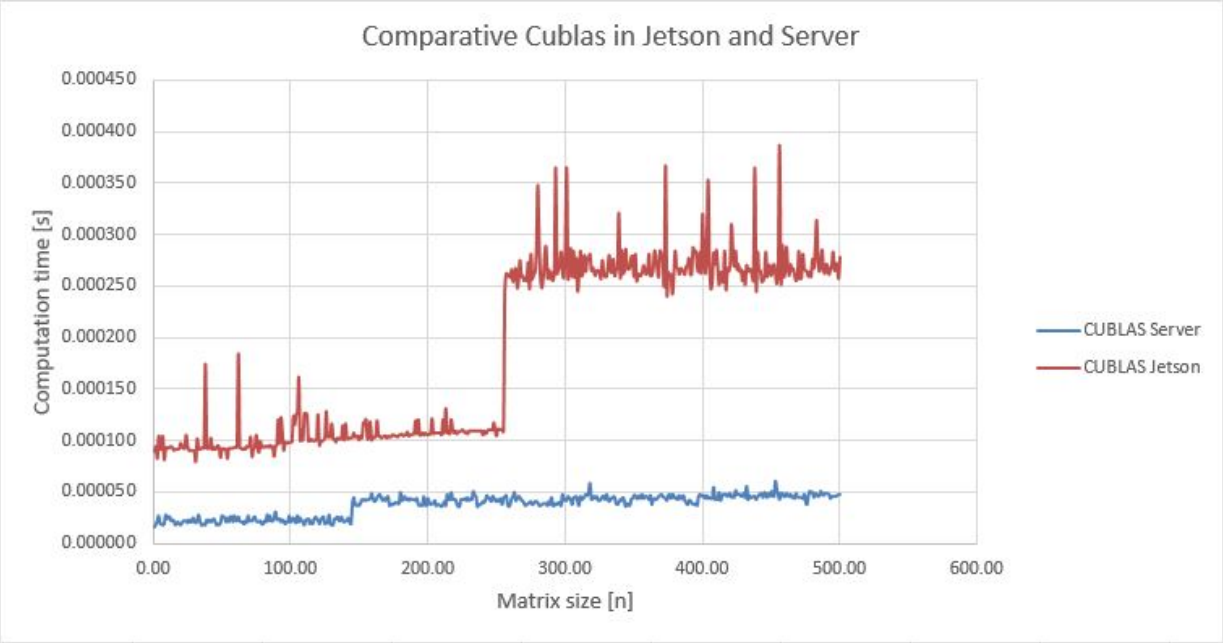

Figure 7: Comparison between the Tesla C2050 and the Jetson TK1 performances.

	Jetson TK1 GK20A	Server Tesla C2050
Total amount of global memory	1892 Mb	2687 Mb
Number of multiprocessors	1	14
CUDA cores (per processor / total)	1 / 192	32 / 448
GPU clock rate	852 MHz	1147 MHz
Memory clock rate	924 MHz	1500 MHz
Memory bus width	64 bit	384 bit
Integrated GPU sharing host memory?	YES	NO

Table 3: Specifications of the Tesla C2050 and the Jetson TK1.

A final test has been conducted on the final GPGPU used in this project, the Jetson TX2. The results can be observed in figure 8. It can be observed that the performance of the TX2, as expected, is higher than the one of the Jetson TK1, already displayed in figure 6.

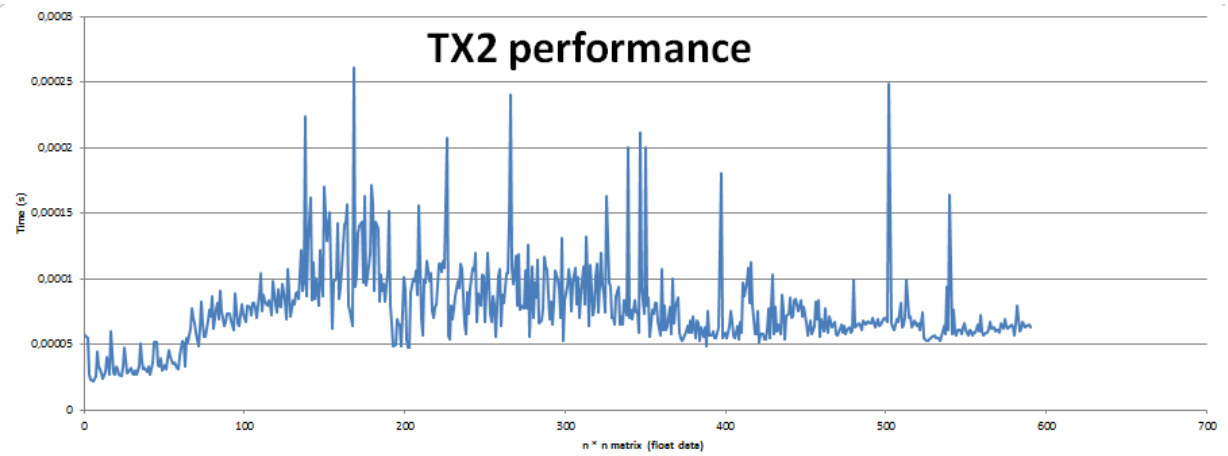

Figure 8: Performance of the Jetson TX2.

5 Obstacle Avoidance and Motion Planning with GPGPU

An overview of the method and the details of its implementation will be given in the following sections.

5.1 Introduction of the method

Following the information gathered in the literature of this project, one method seemed to meet the two requirements that were established for the obstacle avoidance algorithms to be developed for the project: feasibility of implementation and a performance compatible with real time operations. The method also has high potential for a parallel implementation. This method is the Covariant Hamiltonian Open Motion Planning (CHOMP) [14].

A simplified version of the method has been implemented using parallel processing with the CUDA libraries in C/C++ language. The simplification of the method will not affect its own performance, as it regards only the fact that the CHOMP method is designed to take into account multi-joint robots, significantly increasing the complexity and dimensions in the mathematical procedures and algorithms. As the goal of this project is to finally apply this method to vehicle, only one free and single point has to be considered for the obstacle avoidance.

The most fundamental version of this method is based on the gradient descent optimization of an initial trajectory to be as far as possible from every near object at the same time that it keeps the trajectory's curvature within some specified limits. The purpose of the application of a curvature constrain is directly related to its effect on the accelerations experimented by the robot as it follows the trajectory, which will determine the magnitude of the control effort.

These two constrains, the proximity to an object and the curvature of the trajectory, will define the cost function of the optimization method. There is one difference between the two processes that evaluate each constraint: the object cost function can be evaluated a priory, as the objects do not move during the optimization loop, but the trajectory's curvature cost must be evaluated inside the optimization loop, as the trajectory is modified in every iteration of the optimization. This, and the concept of the gradient descent optimization previously mentioned, will be later explained in detail.

For the application of the method in the real world, the space that is analyzed in search of the optimum trajectories must be discretized into finite volume boxes called voxels. The aim of this is to simplify the computations and reduce them to a delimited number of spatial elements.

With all this preliminary information, the general sequence scheme of the avoidance algorithm may be defined as: discretization of the space (with the objects to avoid), com-

putation of the obstacle cost discretized function, computation of the finite gradients over the obstacle cost function and finally the gradient descent optimization loop (including the computation of the discretized curvature and its gradient).

5.2 Description of the algorithm

5.2.1 Spatial discretization

The concept of the discretization of the space composed by obstacles and void space is straight forward: initially, a discretizaion refinement parameter is defined, which defines the number of elements in which each axis of the cartesian coordinate space will be divided. Then, a maximum distance is defined for the axis of the analyzed space. With this parameters, each point of the point cloud obtained from the stereo camera, i.e. an obstacle, will lay into one of the three dimensional bins, voxels, that have been defined.

Now, the process is simple: initialize a three-dimensional matrix with zeros. The size of the matrix will be the number of discretization bins to the power of 3. When a point belongs to a voxel, the value of the matrix should change to 1 if it had a 0 previously. If the value was already a 1, it should not change. Therefore, all the points building up an obstacle can be reduced easily to a bunch of values. Thanks to this the understanding of the obstacle-filled space is simplified to a matrix of ones and zeros.

As a remark, the position of the obstacle is not completely exact in the voxels map. The reason for this is that a single point marks an entire voxel as an obstacle. However, as the volume to be considered obstacle will always be bigger than the real one, the method proves to be safe for the obstacle avoidance purposes.

5.2.2 Distance Transform function

The obstacle cost function is a critical step of the optimization process. It must be properly computed, as it will define the proper avoidance factor for this method: during the optimization it will be evaluated in order to stay far away from the objects.

The obstacle cost function is discretized over the space in the same manner as the voxelized space. The goal is to assign a value to each element of the 3 dimensional matrix that represents how far is this voxel from the closest object to itself.

This is not a new problem: it is commonly referred to as the distance transform, widely used in image processing. Thanks to some research, it was found that, among the existing techniques, few of them are adapted to 3 dimensional cases and even fewer adapted to CUDA [11]. Therefore, based on this 2 dimensional and non parallel methods, a 3 dimensional CUDA-based algorithm is developed. The algorithm work-flow is:

1 - Define three new matrices, same size as the obstacle voxelized matrix. This matrices will contain important information for the method: the distance (in number of voxels) in the x, y and z axis from the current voxel to the closest voxel where there is an object. The main voxelized matrix will store the euclidean distance to the closest object for each element.

2 - Assign and infinite value to all the non obstacle elements and 0 to the obstacle elements. This is done in the 3+1 matrices.

3 - For each voxel (done with CUDA simultaneously), check the values of the voxels situated to the right and left, up and down and front and back form the current voxel.

4 - If during this sequential checking the checked voxel has an euclidean distance lower than the euclidean distance in the current voxel, add 1 to the voxel value in the matrix of the axis where this voxel was found. Example: if the upper voxel has a lower euclidean distance than the current, add 1 to the current voxel value in the Y axis matrix.

5 - Iterate until convergence. The method will update the values in an expanding way starting from the obstacle voxels. The convergence is reached when, for all voxels, the previous and current minimum value is the same, or their difference is really small. Consequently, it could be concluded that all the voxels have the values corresponding to the minimum distance to the closest object.

Some extra features were added to improve the performance of the method, such as the pre-check of the value of the element: if the value is Inf, do not check, the voxel has not been updated yet.

In the figure 9 it can be observed the distance transform obtained for a theoretical setup created in order to test the method. The set is composed by two walls located in the XZ plane and misaligned in the Y axis.

Although the results are as expected, there is still one case that has not been taken into account: when the trajectory goes perpendicularly to an object that blocks its way, there will be no avoidance. The reason for this is that, as it can be observed in 9, all the voxels situated in the proximity of the wall from its upper or lower parts have the same value when compared in the x direction because their distance is the same with respect to the wall. This makes the gradient to be 0 in the X axis and thus, during the optimization the trajectory WPs are not displaced to the right or to the left to avoid the object.

For this reason, a "patch" was applied to the distance transform algorithm. Prior to the normal computation, all the voxels considered an obstacle which are integrated inside an object in any of the three cartesian directions (a wall or any other shape), are subject to a modified distance transform algorithm that computes the distance of each obstacle voxel to the closest voxel that represent the limit of the obstacle in that axis.

The result of this "patch" procedure is that the obstacle voxels have now a descending gradient towards the border of the object in which they are included. As an example, it

can be appreciated in figure 9 that the cost all along the walls is the same from their center to their borders (in X direction). Nevertheless, after applying the already explained patch, we can notice that the center of the object is the most costly voxel of the object, decreasing towards its limits. All this cost is always higher than the non obstacle region of the cost function, so the optimization is assured to avoid the objects, figure 10.

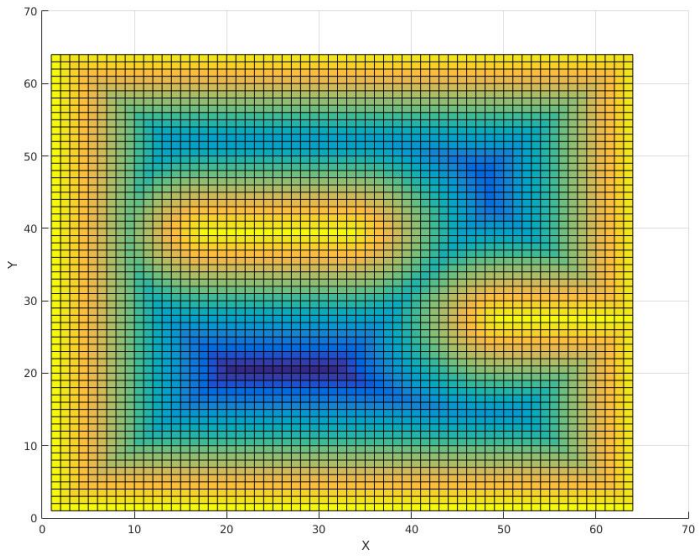

Figure 9: Distance transform of the theoretical example voxelized space. Slice located at Z = 33. Spatial discretization done with a 64x64x64 grid.

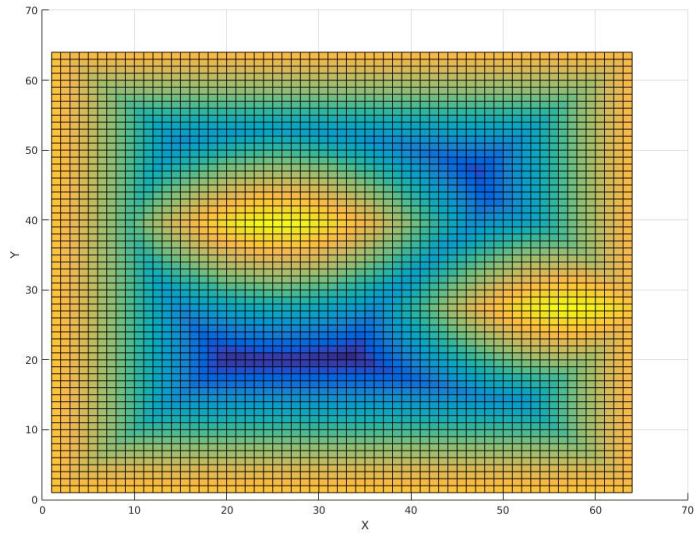

Figure 10: Distance transform of the figure 9 improved by the distance transform patch.

5.2.3 Obstacle cost function

The obstacle cost function is defined simply as the module of the euclidean distance for each voxel. The computation of the obstacle cost function is the most critical step in the algorithm. It will define how well will the optimization behave. Similar techniques to the previously explained patch could be applied in order to obtain more robust solutions and to allow the vehicle take smarter decisions when confronted to complicated obstacle arrangements.This provides the method a high potential for future research in order to improve the results obtained with this algorithm.

5.2.4 Finite gradients

The last step prior to the optimization procedure is the computation of the gradient for each element. This gradient is computed as the derivative of the obstacle cost function in the 3 axis. As the algorithm work only in a discretized space, the derivative will be computed as the finite difference of each element.

For computational optimization purposes, the previously defined X, Y and Z discretized matrices (used in the distance transform algorithm) will be reused to store the value of the derivative in each axis for each element.

The procedure is now simple: compute the finite difference of each element using the values of the obstacle cost function in the neighbour elements of the current element. A 2^{nd} order difference scheme will be used in order to achieve a better defined derivative while maintaining the computational cost low. The 2^{nd} order finite difference, in this example for the X axis, is computed as:

$$dX(x,y,z) = \frac{1}{12} f_{obstacle}(x-2,y,z) + \frac{-2}{3} f_{obstacle}(x-1,y,z) + \\ + \frac{2}{3} f_{obstacle}(x+1,y,z) + \frac{-1}{12} f_{obstacle}(x+2,y,z) \quad (1)$$

For the elements situated just by the border of the discretized space with only one neighbour in the cartesian direction that is being differentiated, a simplified 1^{st} order version of the equation 1 is used.

5.2.5 Gradient descent optimization

Gradient descent optimization, in its most basic application, is explained as a technique to find the maximum or minimum of a function following in each iteration the direction of the gradient. Although this optimization is based on [14], it is actually simpler. Thus, the specific gradient descent technique used so far is the fixed step, consisting on a fixed

step which is followed in the gradient direction to finally converge at the minimum of the function.

On the other hand, the trajectory is defined as collection of way points (WPs) ordered from the current position to the target position. The trajectory will be firstly initialized as a straight line, as it is the most reasonable initial guess of the optimized trajectory. However, the following initialization will be made with respect to the previously computed trajectory. This was a posterior modification that turned out to be necessary in order to let the algorithm avoid the obstacles when it no longer saw them. In the next chapters, these kind of issues that implied a posterior modification are explained.

With the objective of evaluating the gradient from the voxelized space matrices in X, Y and Z (previously computed), some further considerations are needed. The reason for this is that the position in the space of the trajectory WPs is not discretized: they most certainly lie in between the center of several voxels.

The initial technique used was to simply assign the value of the gradient corresponding to the voxel where the WP was. This technique proved to progress correctly but it was difficult to bring to convergence and even evaluate it.

A more developed technique, closer to the continuous space analysis, can be implemented if we consider the fact that each of these points has a total of 8 voxels in its proximity (up, down, front, back, right, left combined in 3 to 3 to form the neighbour voxels to the WP such as: [up,front,right], [down,front,right], ...). Now, a weighted average of the values of these 8 voxels can be used to compute a more accurate gradient for the current voxel. The parameter used to compute the weight of each neighbour voxel is, once again, the euclidean distance from the WP to the center of the voxel.

This later technique turns out to converge faster and properly: when close to the minimum, the average gradient tends to 0.

The computation of the obstacle cost gradient will take into account an important parameter: the safety distance. This is the smallest distance to an object, defined by user, that will be considered as safe for the integrity of the vehicle. When a WP is further than this distance from the closest obstacle, the obstacle gradient will be set to 0.

On the other hand, once the obstacle cost gradient is obtained, the curvature gradient for the current WP configuration is obtained for each WP. Again, this process is done simultaneously for each WP, using the capabilities of CUDA.In order to evaluate the curvature gradient, the cost for the curvature of the trajectory has to be evaluated at the current WP.

The local curvature is evaluated as the module of the projection vector of the vector $(WP(i-1), WP(i))$ over the plane defined by the vector $(WP(i-1), WP(i+1))$. A graphical explanation for this method is provided in figure 11. The curvature gradient vector will be the green one, pushing the WP towards a configuration closer to straight

line, i.e. reducing its curvature.

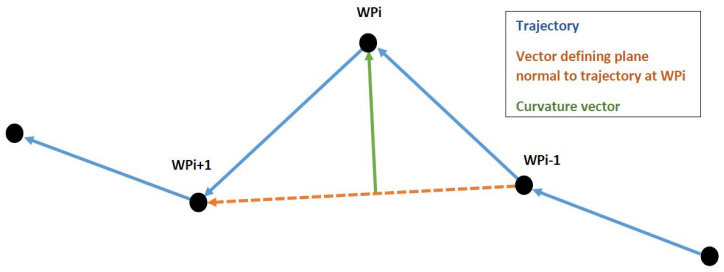

Figure 11: Graphical explanation of the local curvature computation process.

Afterwards, the curvature gradient is added to the obstacle cost gradient with an weighting value multiplied, so the influence of the curvature can be easily modified.

The next step is the fundamental step that guides the optimization of the collection of WP in a trajectory to a successful convergence, in contrast to the optimization of a single point (as all the way points would tend to the minimum). The potential problem is that for several points, the gradient could cause a convergence to the same minimum. Therefore, the solution proposed by [14] is to project the gradient vector onto the plane perpendicular to the trajectory between the previous and the next way points. The component of this projection that would displace the WP in the direction aligned with the trajectory, i.e. towards its neighbour WP, will be neglected. Therefore, the gradients vectors are smoothly modified so that the trajectory won't converge to a single point but, at the same time, these gradients will conserve their property of converging near the minimum value. In figure 12, this process is graphically displayed.

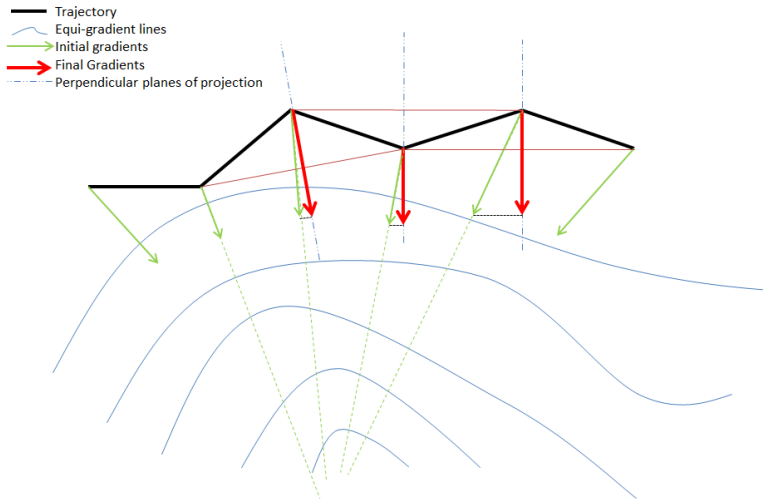

Figure 12: The initial gradients that point to the same minimum are modified with the criteria aforementioned explained in the pursue of an optimum trajectory

However, when using this projection technique a new potential problem arises: if the trajectory has a significant alignment with the gradient field, the module of the perpendicular plane projection would be relatively small, and the effect of the gradient reduced. As this could have a negative effect during the optimization process, the issue has to be solved. To do this, the module of the gradient before the projection is stored and later assigned to the projected gradient. This way, the projection will just define the direction of the new gradient vector and the module will come from the previous one, which is independent from the alignment of the trajectory with the gradient field.

Another gradient has been added to the optimization process: a gradient that will try to keep the distance between way points stable and equal for each segment of the discretized trajectory. This helps the stability of the optimization, as it prevents wrappings of parts of the trajectory.

Furthermore, a fixed WP density strategy was implemented. In order to maintain the density of WPs, i.e. the number of WPs per distance in the trajectory, stabilized; the mean distance between WPs is measured. When this value gets far from the initial WP density value specified in the parameters, a WP will be added or removed. If the mean distance between WPs is higher than the desired value, a WP will be added and if it is lower, a WP will be removed. This also helps the stability of the optimization process, as it is important to keep a sufficiently fine discretization of the trajectory in order to properly describe the avoidance path around the obstacles.

Right before the updating of the WPs, a gradient momentum is implemented. The value of the previous iteration gradient, multiplied by a factor (usually 0.6), is added to the current gradient. This helps the optimization converge in regions of the cost function where the optimization may have an oscillatory behaviour.

The final steps are the convergence checking and the update of the WP coordinates. The convergence criteria is really simple: a value is set to represent the magnitude under which the gradient must be in order to be considered small enough to be close enough to the function minimum. If this value is reached, the optimization loop is finished. If not, iterations continue. This convergence threshold technique has clear drawbacks: the criteria is a quite arbitrary technique, as the optimization may be able to converge only to a higher cost trajectory which is, however, a suitable (collision free) trajectory. To improve this issue, other techniques have been studied to asses the convergence of the optimization. The residuals of the optimization have been plotted in figure 13. It is expected that it could be possible to identify the iteration when the residuals are stabilized at an specific value, so it could be assumed that the optimization has converged. This could be done, for example, computing the standard deviation of a determined number of past iterations residuals.

The update of the values is done every iteration inside the loop and once more, updated to the final WP vector outside the loop when the convergence has been reached.

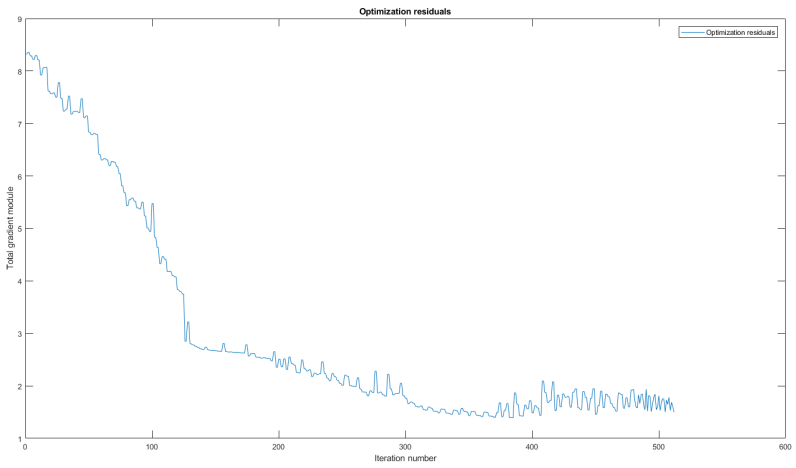

Figure 13: Plot of the residuals during an example optimization procedure.

5.3 Modifiable parameters

Once the algorithm is fully structured, the importance of a good understanding of the input parameters that drive the algorithms performance must be remarked. In this section, the most sensitive of this parameters are mentioned and explained for a better understanding.

- Obstacle and Curvature gradients weighting factors

  There are two weighting factors that will be in constant opposition and closely interrelated. They are the obstacle gradient, which tends to set the way points far away from the obstacles and the curvature factor, which tries to make the curvature of the trajectory as smooth as possible.

  Depending on the weight factor of each agent, the optimization is driven differently, giving more influence to one gradient or the other. A strong obstacle gradient weighting will lead to a faster convergence, as the WPs are pushed to safe configurations faster. However, this will put in danger the stability of the process, as if the momentum is too strong, the optimization may diverge. Then, in opposition, the curvature gradient will help to stabilize the process, but slowing it down.

  The weight factors were considered to strongly influence in the obstacle avoidance algorithm, mainly because they affect the optimization process.

- Convergence criteria

  As explained before, the convergence of the algorithm is now assessed by means of a threshold value of the cost module. When this module is smaller than the threshold, the convergence is reached. This value is therefore a quite sensitive one for the stability of the avoidance. If the value is too small, the optimization may

not converge. If it is too high, the optimization may converge to non feasible or non smooth trajectories.

The value which is being currently used is thirty-fold the gradient step, being this latter 0.05.

- Distance between Way Points

  There exists a parameter that establishes a value for the distance between points in the trajectory. It will determine the number of points in the trajectory. It might be simple to deduce that the higher the number of points (i.e. low distance), the more detailed the trajectory. A fine trajectory helps for a smoother optimization process, but a too fine one may destabilize the algorithm, as it will make undesirable effects, as wrapping of the WPs, more probable to occur.

- Refinement of voxelization

  The refinement of the voxelization affects directly the precision used to describe the obstacles. A higher refinement will help to obtain a smoother, better defined gradient field. However, it has a direct impact on the speed of the algorithm. As this refinement does not affect the robustness of the algorithm (the obstacle swill be always taken into account), its value is usually kept to a relatively coarse value.

6 Theoretical testing

The code was implemented over a theoretical environment generated by computer. In figures 14 and 15 the result of an example optimization procedure over the previously explained theoretical set is shown. The optimization is performed for a trajectory starting at the point (2.5,-4,-4) and target point (2.5,4,4).

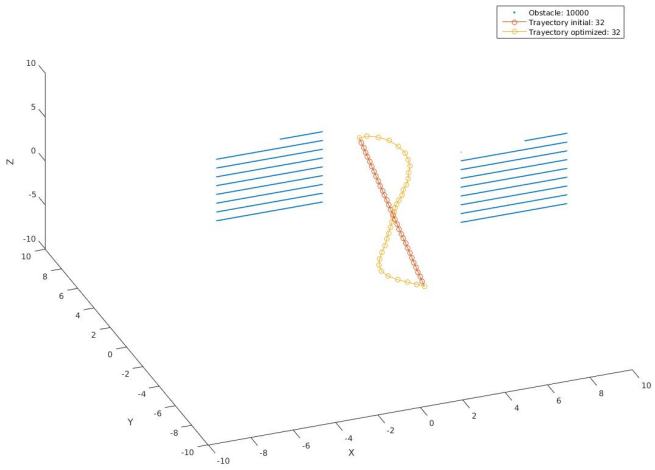

Figure 14: Three dimensional view of the optimization result.

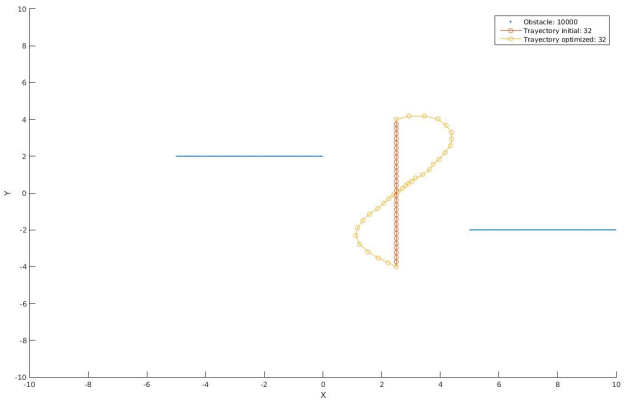

Figure 15: XY plane view of the optimization result.

As a measure of the overall performance of the algorithm, the computational time taken to run the algorithm has been estimated. The results show that a mean of 0,05 seconds is

needed by the current version to compute an obstacle avoidance trajectory. This implies that the algorithm can currently work with a frequency of 60 Hz (or 60 frames per second), which place its performance close to the real time constraint of the current project. All the tests have been done on the Nvidia Jetson TX2.

7 Experimental testing

Now that a obstacle avoidance algorithm has been built and tested in a theoretical environment, experimental tests of the algorithm can be carried out. This chapter will be dedicated to explain the experimental setup and procedures followed in order to validate the algorithm in a real environment.

7.1 Experimental setup

In order to perform the experiments, several systems have been integrated together to allow the flow of information from the sensors to the computational core, and from there to the final controller which will drive the actuators of the vehicle.

7.1.1 Systems integration

The systems can be divided as follows:

- Sensors

  The obstacle detection is made by the stereo camera (ZED), and the data directly read by the avoidance algorithm.

  The Optitrack is used to obtain a real time positioning of the vehicle.

- Vehicle

  Integrates the actuators and structural frame that transports the other systems. The iRobot Create is the vehicle selected for the testing.

- Computational core

  GPGPU. Specifically for these tests, the model Jetson TX2 is the one used.

- Interface protocol

  We will be using ROS (introduced in section 3.4 *Robotic Interfaces*) for interfacing the different parts of the system. In terms of the version, ROS Kinetic was chosen to be installed.

  From a package consisting on the wrapper for ZED Camera [2], one can get significant items such as depth maps, odometry, mapping or point clouds. However, some of this data has to be transferred through other algorithms to be actually worthy. Stereo Labs also provides the .launch files that define the necessary parameters, the required operations regarding transforms in addition to allow the user to use the executable nodes to have access to the data. It is important to remark that this

package, like so many others, needs other packages in order to perform its computations. For instance, there are specific packages to compute the transforms (to have multiple coordinate frames over time) or to analyze messages that come from sensors (in this specific case from a stereo camera). Some examples of messages used in the project are *PointCloud2*, *Ododmetry* or *PoseStamped*; each of them containing different parameters. As their names indicate they contain a Point Cloud, a position (coordinates-quaternions) and an absolute position respectively. Note that even if *PointCloud2* seems to be a point cloud, it is actually the specific type to be treated as a message.

Secondly, a package that deals with point clouds is essential. Basically, its purpose is to obtain the data from the point cloud (number of points, coordinates x, y and z of those points etc.) so that afterwards we can perform actions like visualization, mapping or specific algorithms such as obstacle avoidance.

Finally, there will be the obstacle avoidance algorithm, which will include the necessary tools for publishing and subscribing useful information. This last is not specifically a ROS package, although it is used as if it was (but of course, it is built and run in a different way). We remark that there is a package that deals with the vehicle control.

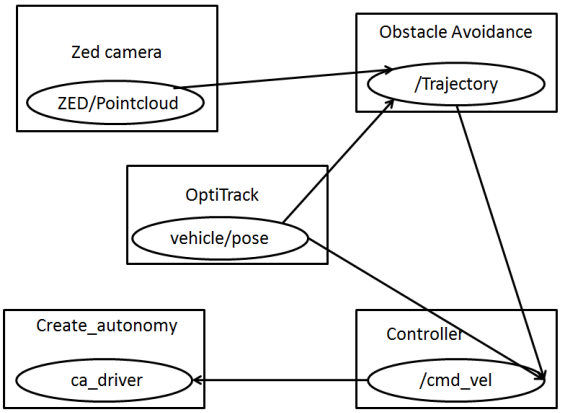

Figure 16: Rqt graph of ROS environment for obstacle avoidance. Case using OptiTrack for the positioning.

We call the some algorithms "packages" as they will be communicating with each other with ROS, but it must be remarked that they are not ROS packages. They are not built as ROS packages but compiled as a C++ codes. However, due to the communication among different nodes, we will call them packages. Figures 16 and 17 show the interactions that should take place while all the nodes are active. Some will publish data on certain topics, from where the subscribers of other nodes will get the data to perform the algorithms. Whereas 16 displays the active nodes when the algorithm takes the absolute position of the vehicle thanks to ROS and OptiTrack,

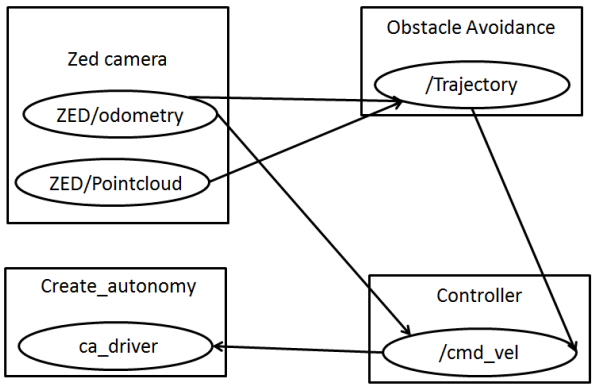

Figure 17: Rqt graph of ROS environment for obstacle avoidance. Case using the camera odometry for the positioning.

figure 17 shows the case in which the position is taken from the odometry provided by the stereo camera. To clarify the picture, see reference [3] to know more about OptiTrack.

WIFI and ssh data transfer protocol is also used for communication with the laptop, which will play the role of ground station.

- Ground station

  A standard laptop is used as ground station, to monitor the data during the tests and give a target position to the algorithm. The target position is the only user given input data, as the system is meant to be autonomous.

7.1.2 Controller

The designed controller is a second-order proportional tracking controller, whose outputs are the linear and the angular velocities. Due to safety reasons, the controller has also a saturator in each variable. Likewise, the input of the controller algorithm is the position to follow. It has to be remarked that this position is not the final target position but the second way point of the trajectory. The reason for this is that, at each frame a new trajectory is computed, thus the path for the robot to follow is slightly different. The vehicle will follow this second WP until there is only one WP left in the trajectory, the final target position. When the way point coincides with the final target (within a tolerance), the controller will stop commanding. On the contrary to the rest of algorithms, all the code for the controller is developed in Phyton.

7.1.3 Software Setup

Software must also be adapted in order to perform the moving vehicle tests. The new version is slightly different because there will be another interaction with the vehicle controller. The main change that had to be made was an adaptation of the reference frames.

The aforementioned Optitrack provides the exact location (within its limits) of the device, but in an absolute frame of the testing room. The avoidance algorithm works in relative frame, the current position being always the origin of the relative frame. It is pretty obvious that these two reference frames are different. Plus, their coordinates do not coincide: whereas the -Z coordinate is the depth in the camera, it is the vertical axis in the OptiTrack reference frame. Due to this, we had to deal with the problem consisting on that some computations were done in relative coordinates, such as the avoidance trajectory; and others, like the controller, were managed in absolute coordinates (OptiTrack). The transformations relative-absolute and vice-versa of different points (target, initial position, way points) were an important tool to eventually get the desired results. One can appreciate the specific functions in the algorithm performing these rotations and translations.

In section 7.1.1 *Systems Integration* we had already presented an alternative which does not have to deal with the just aforementioned drawback. A version of the obstacle avoidance algorithms that only makes use of the positioning of the camera was released and tested, although not implemented in moving vehicle tests.

7.2 Static vehicle testing

Following the satisfactory results obtained in the theoretical testing of the algorithm, the next step has been its testing in a real environment. The first step of this experimental testing is its testing without actually sending a command signal to the vehicle, so the data can be easily analyzed. To do so, the interface with the StereoLabs ZED stereo camera is set and included in the code. Now the real data set can be analyzed and the performance of the algorithm in a real environment may be tested.

The results from this test can be observed in figure 18. In these figure, a point cloud with an obstacle is scanned. In this point cloud, an initial trajectory going through the obstacle to reach a position behind the obstacle is shown in red, and the optimized trajectory avoiding it and any other obstacles is displayed in yellow.

The code was then adapted to limit it to 2 dimensional trajectories. This was done in order to be used with the surface vehicle. The system was again tested on a real environment. The results can be observed in figure 19.

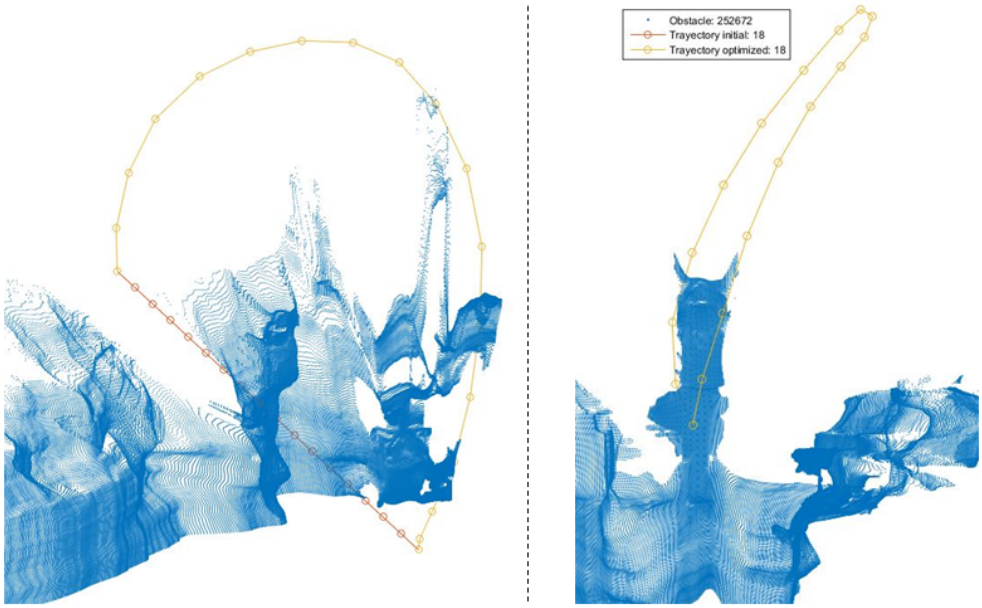

Figure 18: 3D view of the optimization result on a real point cloud.

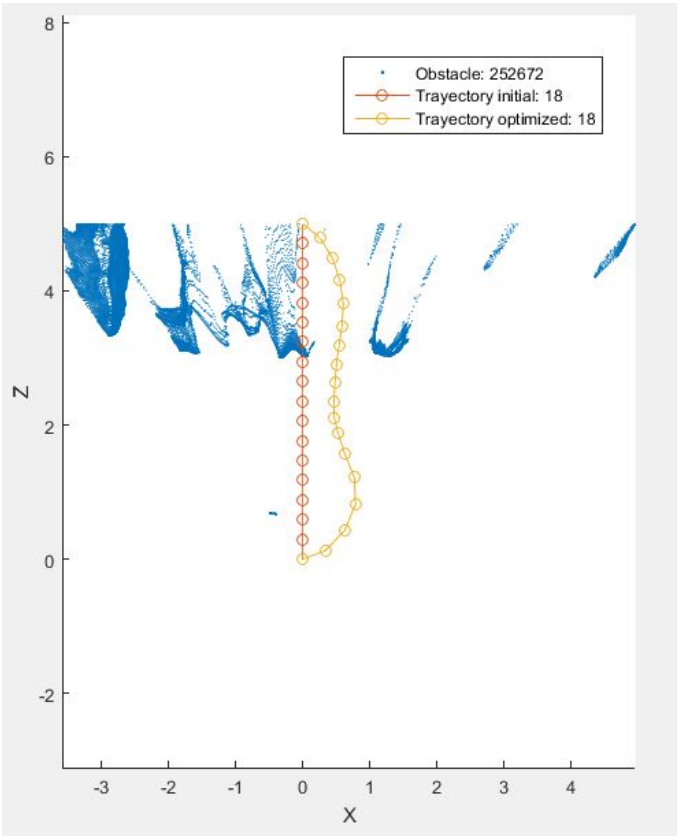

Figure 19: XZ plane view of the optimization result of the 2D adapted code.

7.3 Moving vehicle testing

Once the results of the static vehicle were satisfactory, the overall algorithm was tested on the two-dimensional robots explained in section 2.3 *Robotic Vehicles*. The main difference with respect to the static vehicle tests stems from the need for a controller. This controller aims to give linear and angular velocity commands to the robot so that it moves according to the trajectory. See section 7.1.1 *System integration* for further information about the communication between all the interfaces and algorithms.

Many tests were carried out, with positive results. Of course, not all tests were successful. The failures helped us notice some potential mistakes thus the algorithm had to be modified several times.

This section focuses on presenting the general behaviour of the system, which is more than satisfactory. As expected, the camera is able to detect obstacles and the avoidance algorithm computes a trajectory in around 0.05 (or even less) seconds. This means that the algorithm works at around 60 hertz. The second way point of this trajectory is the one that the controller will try to follow at every frame.

Figures 20 and 21 correspond to two of these tests. A set of columns was established as obstacles. The coloured points are the last five trajectories computed. This makes observable how the trajectories are recomputed and modified as the obstacles are detected. Furthermore, black crosses represent the position of the robot along the actual trajectory that was followed. It can be concluded that the controller was performing suitably as the path always follow the proposed trajectories.

A black arrow has been placed at the initial position of the vehicle in order to indicate the initial yaw orientation of the vehicle. Thanks to it, it can be observed how, for test 1, the vehicle starts already observing the obstacle in front of it, and computes an avoidance trajectory from the beginning. In contrast, in test 2, the robot starts without seeing the obstacle to its left, and the initial straight line trajectory would result in a collision. Once the robot moves and turns, the obstacle is seen and the trajectory is recomputed to avoid the obstacle.

Both tests were satisfactory, as the vehicle correctly assessed the collision hazards properly and on time to avoid the obstacles. Also, thanks to the method used in this project, the resulting trajectories have a minimized curvature that leads to a coherent and smooth control strategy.

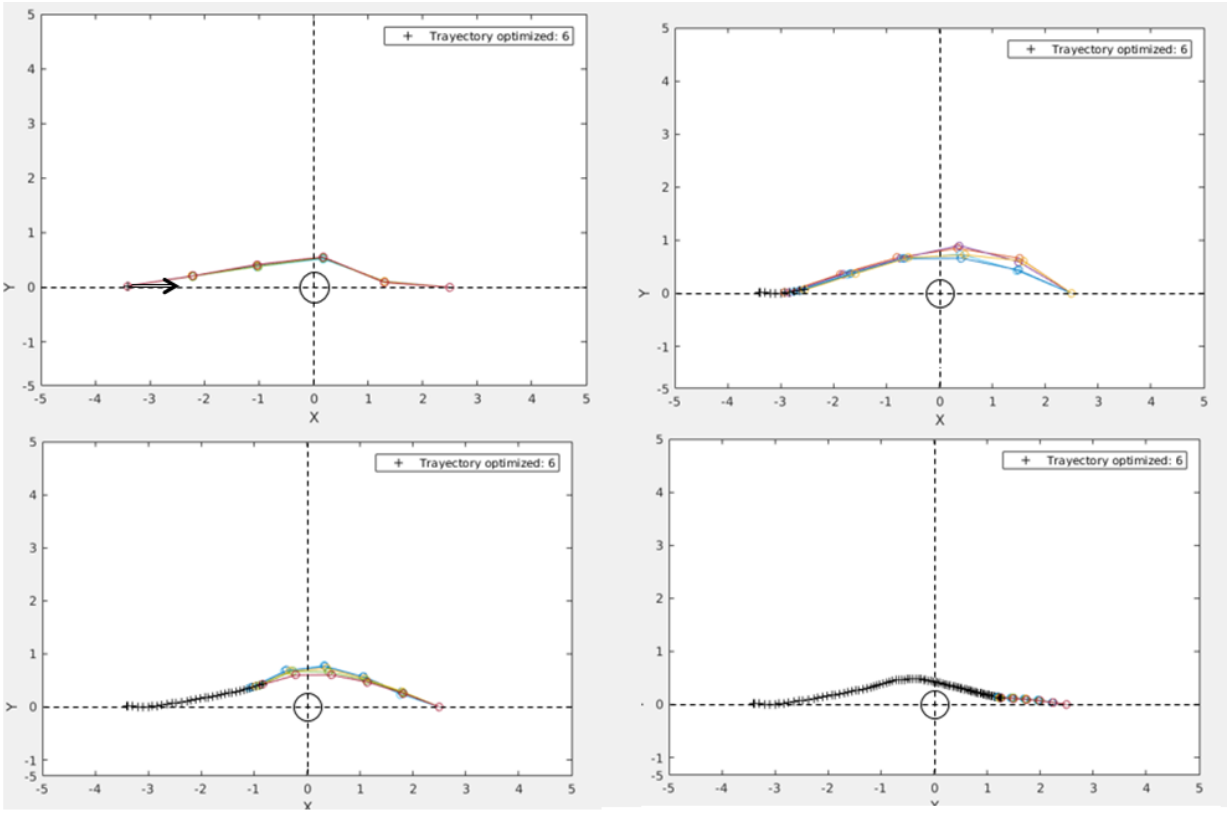

Figure 20: Test 1. The vehicle has to avoid an obstacle placed right in front of it. Satisfactory.

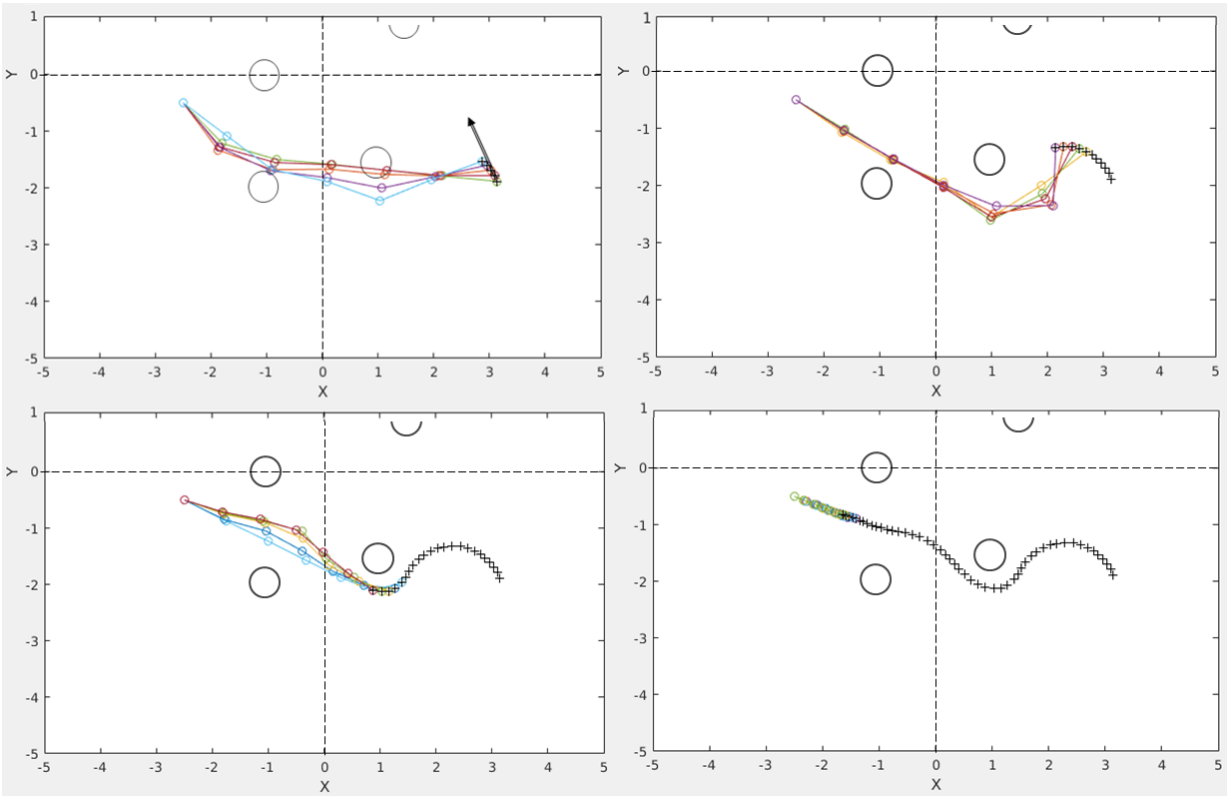

Figure 21: Test 2. In a set of obstacles, the robot manages to arrive to the objective without colliding. Satisfactory.

7.4 Analysis of the results

This section gathers the comments about the carried tests.

The theoretical environment tests were truly satisfactory (see Chapter 6 *Theoretical testing*). As they were the first ones, and the previous step before including the stereo vision device, the main purpose was to obtain a fully reliable results in these tests. These tests were actually helpful to detect mistakes in the main algorithm, even if it was changed afterwards. They defined the basis on which the rest of the tests and developments would rely.

The experimental tests followed a lineal sequence based on adding more and more interfaces to the original algorithm. Firstly, the stereo camera was added in order to obtain a real point cloud for analysis. The tests that checked this computed the trajectories at a velocity of slightly fluctuating around 60 hertz (see figures 18 and 19). Such good results proved that the algorithm could be used in three-dimensional environments with good results. Almost all the tests were satisfactory, with some exceptions whose problems have been analyzed. This problems will be presented in the following sections.

7.4.1 Importance of mapping

When tested in 2D moving vehicles, most of the serious problems arose. Several adjustments had to be made in the algorithm. First appreciable result of these tests (figures 20 and 21) is that the algorithm seems to work according to expectations: it avoids the obstacles and arrives to an objective.

On the contrary, there is still the problem of not taking into account the obstacles and features that are not anymore inside the vision field of the stereo camera (due to the lack of mapping). In figure 22, test 1, this phenomenon is clearly striking. Whereas in the first image we observe how the trajectory is computed perfectly to avoid the column, in the second image the trajectory appears to have changed instead, and be closer to it. What really happened there is that the trajectory was recomputed and the camera wasn't no longer catching sight of the column, so the new trajectory does not take it into consideration. Then, it passes closer to it.

Stereo Labs provides a Software Development Kit, or SDK, from which you can extract some functions regarding the ZED stereo camera functionalities [15]. Indeed, as the algorithm needs to extract the point cloud, these functions were truly useful. In fact, apart from a depth map, it is possible to obtain a mapping that regards not only the current vision field of the camera, but also the past points that it saw.

However, according to Stereo Labs, the mapping functions (those that remember the past points) are not suitable for obstacle detection and avoidance. That is why it is not used in the present algorithm. Consequently, the algorithm performs with the point cloud seen

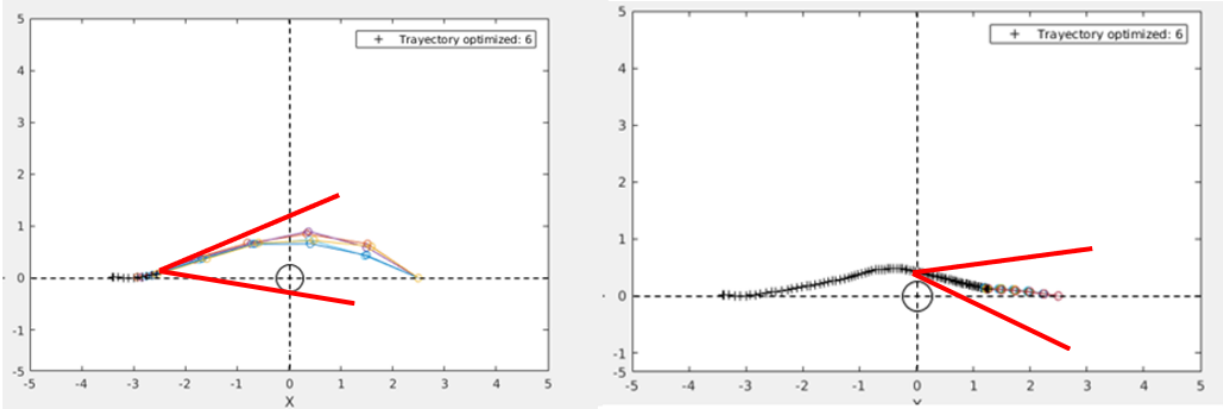

Figure 22: Test 1. Detail of the field vision of the camera. When it does not see the column anymore, trajectory passes closer to it, although thanks to initialization from the previous trajectories, there is no collision.

at each moment, "forgetting" the previous point clouds.

Even if this seems catastrophic, it is solvable (We see that the test resulted in satisfying outcome). There exist some possible solutions. The main one implies an initialization of the trajectory with base on the previous one. In the next point, this is better explained.

7.4.2 Trajectory initialization

Trajectories were originally initialized in a straight line. This was a good idea that had some negative aspects instead. Why?

If a trajectory is computed to avoid an obstacle, the vehicle would move following the trajectory. However, the trajectory is initialized in a straight line at each iteration. Then, if the camera stopped watching the obstacle, the new trajectory (initialized from a straight line) wouldn't take into consideration the obstacle, as it forgot it. This could cause a crash onto the obstacles.

We can see that this method of initialization combined with the not use of mapping has no positive results. In order to partially solve this, the initialization method was changed. The trajectories (with the exception of the first one) are now initialized with the previously computed one. As a consequence, if the camera stopped observing an obstacle, the new trajectory (which does not see the obstacle, so it does not take it into account) would be computed with base on the previous one; which was computed "watching" the obstacle. Therefore, it would implicitly avoid it.

We can conclude that this is not the best solution (which would be to take into account both the mapping and the previous trajectory), but it satisfies the expectations. See chapter 9 *Future Work* for further information about the definitive solution.

All the fully successful tests were made with static obstacles with a specific size. Then,

the nature of the tests was slightly varied in order to test the behaviour of the system in different situations.

7.4.3 Stereo vision limitations

As mentioned before, stereo vision is based in the comparison between the two video streams obtained from the cameras, and then finding the correspondence between pixels in both images. Finding this correspondent pixel can be a problem if the object being analyzed has a uniformly illuminated, coloured surface. This will produce result in big set of pixels with the same values, and affect negatively the depth sensing.

According to this, the system reacted better to surfaces with imperfections than to completely polished surfaces. It was observed also that transparent objects were, evidently, not detected.

The algorithms used for the depth sensing were provided by Stereolabs in their SDK. These algorithms, although constantly being improved by the researcher community, still have some minor performance issues. Consequently, the point cloud obtained by the stereo camera had a "vibration", a slight difference from one frame to the other. Also, the precision of the positioning of the points can be improved, as some flat surfaces, as the floor of the testing room used in this project, were detected as a slightly undulated plane.

Also, for some scenarios, the range limitations of the camera (0.7m - 20m) may be a problem for obstacle avoidance.

7.4.4 Moving obstacles

Other tests were performed putting obstacles in front of the vehicle while it was moving. The algorithm reacted really well. As it works in real time, moving obstacles were not supposed to be a problem at first sight at it was demonstrated that these assumptions were right. Of course, this is applicable as long as the moving obstacles are within the range of vision of the camera (0.7m - 20 m). Obstacles that appear really near to the camera could not be detected. Particularly, these test were performed for the department of the Office National d'Etudes et de Recherches Aérospatiales (ONERA) in Toulouse.

7.4.5 Avoidance algorithm limitations

Finally, and as it was expected, some limitations were tracked and observed in the avoidance algorithm.

- Some minor issues were observed regarding the convergence of the of the optimization loop. These depended on the arrangement of the obstacles in the analyzed scene.

The redefinition of the convergence criteria has might be the most suitable solution for this issue.

- Some configurations of obstacles result in obstacle cost functions that do not lead to a proper avoidance trajectory. The main example of this issue are flat walls that extend from side to side of the analyzed space. In this case, the avoidance will be performed in perpendicular to the wall, but not around it. This would result in a collision. Another important issue is when two obstacles are placed at a distance smaller to twice the safety distance between them. The algorithm may find a local minimum between them and consider it a feasible solution.

  All these problems, and some others that have not yet been spotted can most certainly be solved by means of the implementation of smarter, more sophisticated distance transform and obstacle cost algorithms. This provides the technique presented in this project an interesting potential for future, improved implementations. See chapter 9 *Future Work* for further information.

8 Conclusions

First point to be treated is the clearly remarkable capability of the GPGPUs and the CUDA library. None of this work could have been done without the computational power that this type of architecture provides. As a simple example related to this work, the CUDA kernel (function) in charge of computing the discretization of the voxelized space could do it in only one computation: each thread being analyzed practically simultaneously. In a standard CPU architecture this would have need to be done one by one in a row. Considering the three-dimensional matrices analyses, a computation in CPU would have yielded a performance absolutely non acceptable for real time applications.

Secondly, regarding the results of the tests, the system was proved to work according to expectations. The real time capabilities of the system are more than suitable for the purposes explained at the beginning of this report. Moreover, thanks to the consideration of modelling the trajectories heeding the curvature of the trajectory in addition to the obstacle avoidance gradients, the resulting trajectories are both smooth and far enough from obstacles.

The main objective of this project has been covered. An obstacle avoidance algorithm capable of running in real time, and embedded on board of a lightweight vehicle has been designed. The robustness of the algorithm, even though it has been satisfactory and sufficient for the tests, can be improved in order to meet real life applications standards.

All in all, this project has demonstrated that the combination of stereo vision devices with parallel computation can yield interesting results for the purpose of real time on board obstacle detection and avoidance. This suggests that more research could be done in this field which, improved and combined with other technologies, might lead the way to develop more robust autonomous vehicles.

9 Future work

The present section is considered to be truly necessary so that next generations can continue working on this project. As it was appreciated all along the development of the present project, its possibilities expand exponentially. This made it impossible to continue studying all the aspects concerning the project. Then it was decided to continue in the path of obstacle avoidance, leaving aside some other aspects (some important, others less significant) that could be perfectly object of study for the continuation of this project.

There were secondary objectives concerning new functionalities for the systems. Specifically, these functionalities were automatic landing and pattern recognition and tracking (see section 1.2 *Objectives*). In order to satisfactorily fulfill these objectives you have first to make the system able to be fully autonomous, this implying obstacle avoidance. Plus, it will be always necessary to have functions regarding voxelization of the space, detection of obstacles etc. These fundamental functions have been already developed and can be used in the future, when specific algorithms are designed to meet secondary goals requirements.

The experimental tests were performed in 2D because of two reasons. First of all, the unavailability of the on board version of the GPGPU made it impossible to use it in a drone. Secondly, the algorithm had to be fully tested. This meaning that the tests had to be first performed in a harmful vehicle, for the good of the devices integrity. Furthermore in the future, when there are available and suitable devices and more tests have been carried out, it could be implemented in tridimensional environment. The tridimensional algoritm has been also tested, demonstrating its good theoretical performance.

Nevertheless, there is an aspect of the project that could give rise to a new project. One of the interesting aspects of the project was the use of a GPGPU as computing device, making the parallel operations possible. It is sure that in the present project we did not squeeze the whole CUDA's capacity. However, it is possible that a deeper study in the use of CUDA [11] could improve the performance of the algorithm, as well as the use of the memory. Furthermore, it is suggested that this topic (maybe applied to the current algorithm) is studied to use its capabilities to the full. This should increase the performance in terms of time computation and performance.

In order to improve the robustness of this method, the convergence to local optima which represent non-feasible solutions has to be tackled. This could be solved by improving the way the distance transform and obstacle functions are computed. Smarter, more sophisticated algorithms that assure an appropriate distribution of the gradient fields could have an extremely beneficial impact to the method.

It has been emphasized several times that the performance and behaviour of the system could be significantly improved if an implementation of a mapping algorithm was added. Even if some simple mapping functions were tested, the results were not good enough to implement it into the presented algorithm. Not only would it be interesting but also

truly necessary to develop a real time mapping algorithm. This would let the system to be able to "remember" all the past points in the point cloud thus, the trajectories would be computed taking into consideration obstacles that are still there, even if they are not observed by the stereo camera anymore.

The mapping algorithm could imply a totally deep study in SLAM [16] and its features, taking into consideration moving obstacles, or Artificial Intelligence for the system to learn which is the best decision to take in each situation.

As a final comment on more advanced techniques that could help the project, Artificial Intelligence techniques, as machine learning or neural networks, could be applied to the automatic selection of the parameters of the algorithm. This could help to have the best parametrization for every different scenario to analyze. This advanced techniques could also be applied to the computation of the Distance transform and obstacle cost function, in order to provide the system decision making capabilities in case of complicated obstacle arrangements.

References

[1] "Developing for the intel realsense™ camera (r200)." `https://software.intel.com/en-us/realsense/r200camera`. Accessed: 03-2017.

[2] "Zed stereo camera." `https://www.stereolabs.com/zed/`. Accessed: 03-2018.

[3] "Optitrack." `http://http://optitrack.com/`. Accessed: 03-2018.

[4] "zfas tttech project: Advanced driver assistance system platform for audi." `https://www.tttech.com/markets/automotive/projects-references/audi-zfas/`. Accessed: 03-2017.

[5] C. Georgoulas, L. Kotoulas, G. C. Sirakoulis, I. Andreadis, and A. Gasteratos, "Real-time disparity map computation module," *Microprocessors and Microsystems*, vol. 32, no. 3, pp. 159–170, 2008.

[6] F. M. Wahl, "A coded light approach for depth map acquisition," in *Mustererkennung 1986*, pp. 12–17, Springer, 1986.

[7] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.-M. Frahm, R. Yang, D. Nistér, and M. Pollefeys, "Real-time visibility-based fusion of depth maps," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[8] "Opencv." `http://opencv.org/`. Accessed: 01-2017.

[9] M. Nieuwenhuisen and S. Behnke, "3d planning and trajectory optimization for real-time generation of smooth mav trajectories," in *Mobile Robots (ECMR), 2015 European Conference on*, pp. 1–7, IEEE, 2015.

[10] A. J. Barry, *High-speed autonomous obstacle avoidance with pushbroom stereo*. PhD thesis, Massachusetts Institute of Technology, 2016.

[11] C. Nvidia, "Nvidia cuda c programming guide," *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.

[12] Nvidia, "Cublas library, user guide," *Nvidia Corporation*, 2016.

[13] "Ros wiki." `http://wiki.ros.org/ROS`. Accessed: 03-2018.

[14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 489–494, IEEE, 2009.

[15] "Using the zed camera with ros." `https://www.stereolabs.com/blog/index.php/2015/09/07/use-your-zed-camera-with-ros/`. Accessed: 04-2017.

[16] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*, pp. 834–849, Springer, 2014.

List of Figures