Recognition and Estimation of Obstacles Trajectories in Scale Driving Vehicles

José Guilherme Penas Silvério
jose.silverio@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

June 2017

Abstract — The autonomous robot navigation, avoiding collisions, in an unknown environment is an active area of research. This recognition usually relies on sensors implemented in the system itself, such as cameras or lasers. The system studied in this dissertation is a scale radio controlled vehicle in which it was intended to implement a single camera and to develop software that would allow the system (RC and sensor) to move autonomously.

Since depth recognition through monocular vision is mathematically impossible, a method has been developed that takes advantage of the RC movement and of the capture of multiple images, to detect obstacles and to calculate distances to them. This method involves the pairing of points of interest in the captured images and the displacement relations of these points for the segmentation of obstacles and their mapping. A path planning method (A* algorithm) was implemented after mapping the location of obstacles in space, to define the system paths on the map.

The results in 30 performed tests show that the implemented algorithm causes the system to collide with some obstacle 2.9% of the times when the map is updated. These results are due to small differences between the expected position and the measured position of the system at each time, which directly influence the clustering processes which consequently leads to a poor mapping of the environment.

***Keywords*—Autonomous System, Monocular Vision, Interest Points, Segmentation, Path Planning**

I. Introduction

A. Motivation

The last decade has seen an expansion in the development of autonomous driving vehicles. It seems certain that this expansion will expand your impact on society [1]. The evolution of the means of transport has been mainly to improve security conditions, reduce energy consumption and decrease of the gaseous pollutants emitted by transport. Research in these areas has led to an increase of interest in the development of autonomous vehicles [2]. Like the transport of people and goods, also in reconnaissance vehicles as drones or ground vehicles have been wide scientific development. It is in this framework that is this dissertation.

This work was carried out with the objective of developing software to implement on a sensor to be applied in a scale RC model, so that it could move autonomously in space. For this purpose, the system would have to interpret the information captured by the sensor and create a map where it could move. The sensor to be coupled to the RC is a camera. However, with only one camera, the identification of objects and calculation of distances to them is not a trivial problem, so the first part of the dissertation rests on the method developed to solve this problem. What is proposed is a way of getting around the ambiguities that arise from the capture of an image alone, with the capture of multiple images in distinct positions of the system. This procedure is generally known in research literature as getting "structure from motion". The movement of the system in space is studied next, introducing the principles of path planning and the algorithm developed for the overall system to work. followed by a section of system identification where the parameters of motion of the RC are studied. Finally, the results of the implementation are stated and discussed, finishing with conclusions and future work.

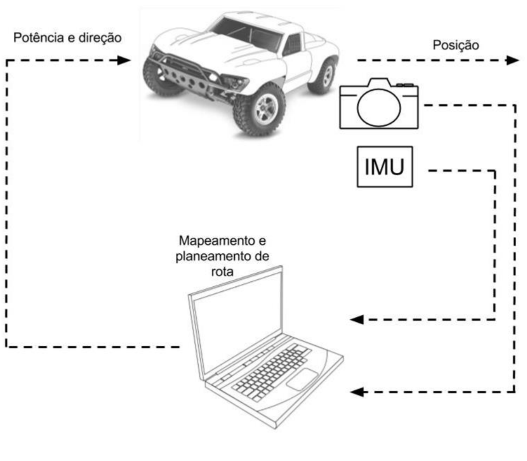

Fig. 1 – System

The RC is a 1/18 scale model in which previous software [3] was already developed for its remote control through Matlab. In the RC itself there is a Raspberry Pi 3 Model B which deals with the communication between the Matlab commands sent by a remote computer and the power and direction controller of the RC. There is also an IMU attached to the raspberry Pi present in the RC, equipped with accelerometer, magnetometer and gyroscope.

The majority of the code developed in this work was written for Matlab. Only the communication with the RC had a python code component.

B. Related work

The human being and most mammals have in their vision their most important sense for the perception of the reality that surrounds them. It is amazing how easily the brain interprets the three-dimensional world through our eyes. Without apparent effort, it is possible to identify objects, count the number of objects, have depth notion or distinguish the various colors and transparencies [4]. The area of research called Computational Vision aims at the approximation to the vision capabilities of living beings with cameras and computational means.

Within the area of computational vision, one of the main challenges is the recovery of the three-dimensionality of the world from an image. Among the various image-based methods for measuring object distances are, stereo vision (two cameras), a single camera, structured light, time of flight cameras, among others. Much of the existing literature addresses this subject with binocular vision (stereo) [5] [6] [7], since it is a relatively accurate way of calculating depth. However, due to the need to obtain and process two images simultaneously, it becomes a more computationally loaded method. In the field of robotics there are already simpler solutions with only one camera, so the use of stereo vision is now more limited. Time-of-flight cameras allow you to obtain information about distances to objects by measuring how long the light takes to travel the distance from the camera to the subject and back to the camera. However, these cameras still have prohibitive costs and their use is limited to specific applications.

The structured light method, also reserved for specific applications, has become very popular, in particular, through the Microsoft Kinect sensor.

Despite that, in this project it was intended to implement a method that uses only one camera, since the space available in the RC to couple sensors is not much, which leads to the need to think of simpler sensors, and because it presents an interesting challenge to attempt to find a method to circumvent the fact that it is mathematically impossible to extract depth from a single image.

Existing works related to depth perception using a camera only focus on machine learning algorithms [9], or adding a laser or lateral movement of the camera to obtain two lateral perspectives and adapt the stereo vision [6] [8].

The challenge of the work is to use the movement of the system (RC and sensor) to help with the identification and location of obstacles.

After defining a method of locating the obstacles by the camera, it is necessary to formulate a method that creates a path around the obstacles encountered. This path is defined by a path planning algorithm. It is important to note that the path created by the route planning algorithm will have to be updated as the camera gets more information about the space. The path planning algorithms have as principle the displacement of an initial configuration of the system to a final configuration. For the definition of the algorithm to be used, it is important to define whether the algorithm will be introduced in an offline or online system and what the dimensionality of the space in which the system can move. These algorithms are optimization processes, since, in most cases, the paths found are optimal or sub-optimal with respect to time, distance or energy.

The robotic path planning problem is a problem that continues to be studied and for which many solutions have already been proposed [8] and several different types of methods applied, such as C space methods, potential field methods or neural networks [9]. For the problem present in this work, given that it is a problem where the configuration space in which the system can move is two-dimensional, an algorithm based on a C-space method was used, since it would not overload the system computationally. In order to solve the problem, it was mainly used [10] [11], where Steven M. LaValle presents an extensive investigation and explanation of methods related to path planning.

II. Image Processing

C. Pinhole model

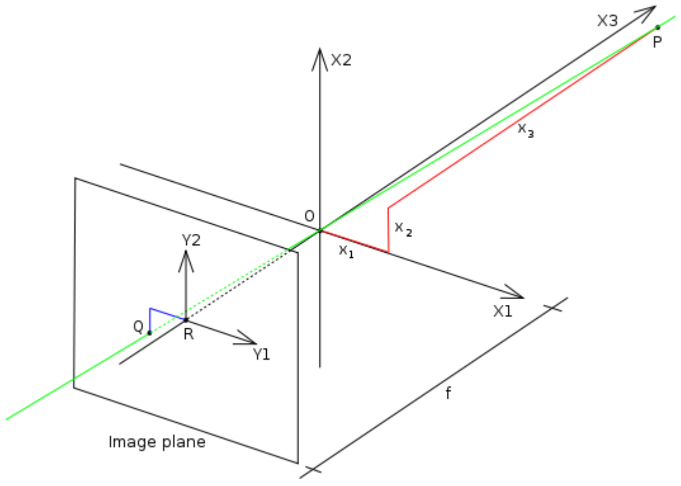

Fig. 2 – Pinhole model [12]

The pinhole model uses triangle similarity to mathematically match a point in an image to a physical point in the three-dimensional real world. In the previous figure is represented a schematic representation of the pinhole model. Three orthogonal axes (X1, X2 and X3) are represented where the point of interception of those axes corresponds to the optical center of the camera (O). The X3 axis is called the main axis of the camera, and the plane represented, perpendicular to the main plane at a distance f (focal length), is called the focal plane.

The point P has coordinates (x1, x2, x3) and is projected in the focal plane, where it is identified by the point Q whose coordinates in the local ordinate axes (Y1, Y2) are (y1, y2).

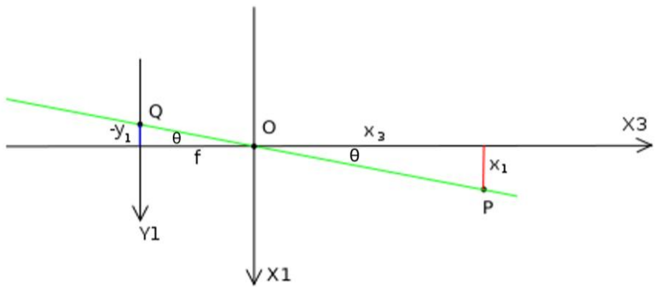

Fig. 3 – Perpendicular plane to X2

The similarity of triangles mentioned above is explicit in the figure, and some important relations can be taken from it.

$$\tan(\theta_1) = \frac{x_1}{x_3} \quad (1)$$

$$\tan(\theta_1) = \frac{-y_1}{f} \quad (2)$$

Similarly, for a projection on the X1 axis,

$$\tan(\theta_2) = \frac{x_2}{x_3} \quad (3)$$

$$\tan(\theta_2) = \frac{-y_2}{f} \quad (4)$$

Note: for all the equations is important to notice that the focal length is constant throughout.

With two images captured after linear movement of the camera there are no ambiguities regarding the positions of the elements in the environment. It is based on this premise that the method of calculation of distances to objects proposed in this work is sustained.

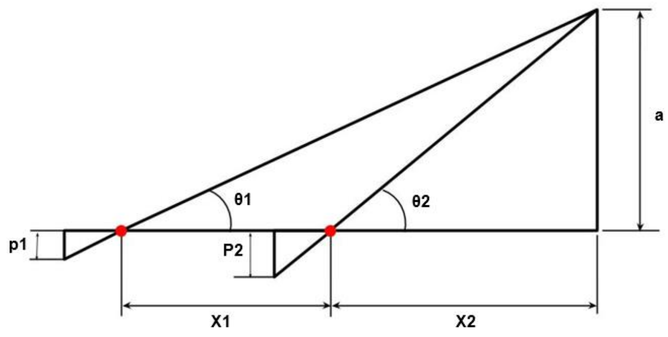

Fig. 4 – trigonometric deductions

In Fig. 4, the red dots represent the camera positions for capturing the images, p_1 and p_2 are the pixel values corresponding, on the images, to the distance a.

$$\tan(\theta_1) = \frac{a}{x1 + x2} \quad (5)$$

$$\tan(\theta_2) = \frac{a}{x2} \quad (6)$$

Isolating and solving both of the above equations in order to a, we arrive at the following results:

$$\mathrm{a} = (x1 + x2)\tan(\theta_1) \quad (7)$$

$$\mathrm{a} = \mathrm{x2}\tan(\theta_2) \quad (8)$$

Equating equations (7) and (8) and isolating x_2,

$$x2 = \frac{x1\, tan(\theta_1)}{tan(\theta_2) - tan(\theta_1)} \quad (9)$$

Developing (2) an (9),

$$x2 = \frac{x1\, p1}{p2 - p1} \quad (10)$$

Eq. (10) only depends on known variables, the distances measured in pixels in the two image captures (p1 and p2) and the distance traveled by the camera between these two points (x1), so this is the relation found to calculate distances to obstacles in this project.

D. Interest Points and descriptors

Finding correspondences between two images referring to the same scenario is an area studied exhaustively by the field of computer vision and that has several applications, such as: camera calibration, 3D reconstruction or identification of objects [13]. This process is called feature detection and feature matching.

From all the existing methods, in this project it was necessary to use a scale invariant and rotation invariant one, so, two of the most common were studied, SIFT [14] (Scale-Invariant Feature Transform) and SURF [15] (Speeded Up Robust Features), which is a faster, slightly different method than SIFT, and so, the SURF method ended up being the one implemented.

The SIFT method uses difference of Gaussian as an approximation to Laplacian of Gaussian (LoG) for the detection of interest points, whereas the SURF method uses a box filter based on the Hessian matrix, also as an approximation to LoG. The advantage of using a box filter in relation to methods for detecting interest points in other approaches is due to the possibility of using the integral image, so that the time spent in detecting points of interest does not depend on the size of the filter, whereas in the case of Gaussian filters, the time required to perform the same process is in the order of magnitude of the image size multiplied by the size of the filter. The SURF method subdivides the neighborhood of each point of interest into squares of 20x20 pixels, further subdivided into sets of 4x4 pixels and in each of these sets passes a Haar filter. For each subdivision of 4x4 pixels, values are calculated for the sum of the filter responses and for the filter responses module, in the horizontal and vertical direction, $v = \left(\sum d_x, \sum d_y, \sum|d_x|, \sum|d_y|\right)$. The vector of characteristics of this method then has only 64 dimensions, that is, half the size, when compared to the SIFT method.

E. DBSCAN

The data to be grouped in clusters comes from the interest points found in the images, aiming to group the points belonging to each different object, in different clusters as well. Thus, it is assumed that the collected points of each object are grouped next to each other in agglomerate. Knowing that the clustering algorithm used shouldn't take a lot of time to run and that the number of clusters to form is not defined beforehand, several clustering algorithms were thought of arriving at the DBSCAN algorithm.

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.

The algorithm requires 2 parameters: ε, maximum possible radius for the neighborhood of the point p, and the minimum number of points required to form a cluster (minPts). It starts at a random point p and adds points from its neighborhood. If it contains enough points to satisfy the conditions imposed, a cluster begins. Otherwise, this point is classified as noise.

If a point belongs to a cluster, its entire neighborhood ε also belongs to that cluster. When all points in a given cluster are defined, the algorithm moves to an unvisited point and follows the same procedure.

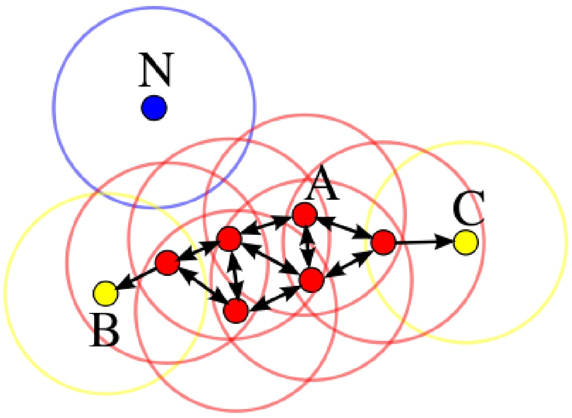

Fig. 5 – DBSCAN method [16]

In Fig. 5, some points of a data group are represented. It was defined that the minimum number of points needed to form a cluster would be minPts = 4 and that the maximum radius of the neighborhood of each point would be represented by the circles around each point. The point A, together with all the red dots represented, belong to the cluster core, since all of them have, in their neighborhood ε, at least 4 points (including the point itself). Points B and C, although they do not have 4 points in their vicinity, are in the vicinity of nuclear points, so they also belong to the cluster formed by red dots. The point N is a point of noise, because in its neighborhood there is no point belonging to the nucleus of the cluster.

F. Distance calculations

Before being able to apply (10), the data from the images must be processed. This starts with identifying the different obstacles that need to be mapped.

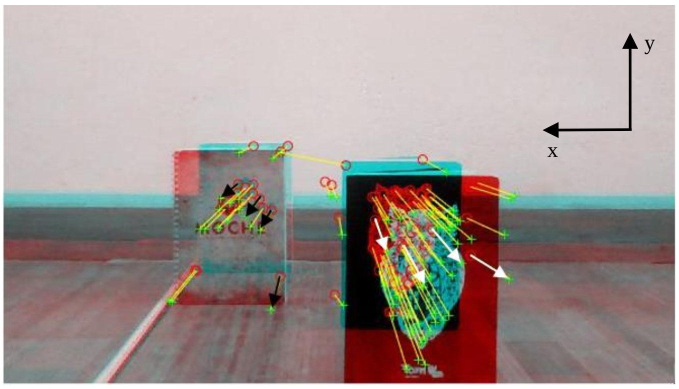

Fig. 6 - SURF matching

Fig. 6 illustrates the matching between two different images taken 10 cm apart in a straight line. The vectors formed with the points matched will be used to identify the different obstacles using the clustering algorithm. As the object on the right is closer than the one on the left, it is easy to notice that the vectors formed from the points on the right have bigger magnitudes, and also, because of the different positions, opposing directions.

After the clustering of the points in distinct groups, the differences between distances of points in the first and the second image taken are used to calculate the positions of the obstacles in space using (10).

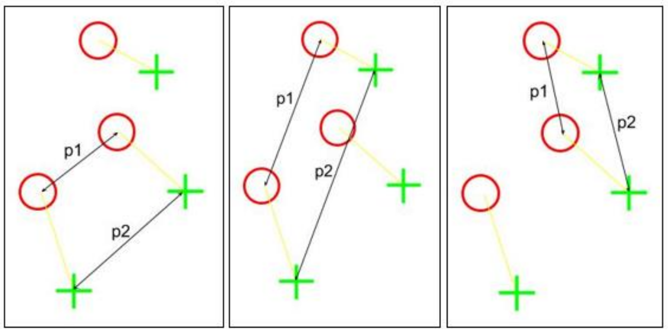

Fig. 7 – Iterations for mapping calculations

In Fig. 7, the points in green represent the interest points detected on the second image taken, and the red points, on the first. The method compares all the possible values of p_1 and p_2 to get better results regarding distance measuring. After all the possible values for the distances are calculated, for all the clusters, the mean value of those distances is found and that is the value used for the longitudinal distance mapping the obstacle.

As for the lateral distance of the obstacles, a similar approach is taken, just with a slight difference, that has to do with the final measure needed. For the problem before, the unknown value from the triangle similarity seen in Fig. 4 was x_2, but for lateral distance measuring, the unknow part is the distance between the center of the image and the obstacle. Having calculated the longitudinal distance to the obstacles already, the pinhole method gives the required relations to calculate the lateral distance as well. Equating (1) and (2),

$$\frac{p}{f} = \frac{a}{b} \tag{11}$$

Unlike the case before, for the lateral calculation, there's the need to know the focal length of the camera in use.

III. Path Planning

G. Problem Formulation

The formulation of the problem is done using state space models. The basic principle is that each specific situation is defined by a state x, and that the set of all possible states is called X. Each state x can undergo changes when an action u is applied, thus producing a new state x '. This transformation can be defined by a function f, called the state transition function. Thus, the discrete state transition equation can be defined as follows:

$$\mathrm{x}' = \mathrm{f}(\mathrm{x}, \mathrm{u}) \tag{12}$$

Similarly to what has been defined previously for the set of states X, U (x) defines the space of actions for each state x, which represents the set of all possible actions. It should be noted that for x, x' ∈ X, U (x) and U (x') are not necessarily different. The same action can be applied in different states. In this way, it is convenient to define the space U corresponding to all possible actions on all states.

$$\mathrm{U} = \bigcup_{x \in \mathrm{X}} U(x) \tag{13}$$

Part of the formulation problem involves defining a set of objective states Xg ∈ X. The purpose of a planning algorithm is to find a finite sequence of actions that transforms the initial state x_1 into a final state Xg.

Summarizing, the formulation of a discrete planning problem:

- Define a state space X, which is a set of finite states.
- For each state x ∈ X, define a finite space of actions U (x).
- Define a state transition function f that produces a state x '∈ X for each state x ∈ X and u ∈ U (x). The state transition equation is given by x '= f (x, u).
- Define an initial state x1 ∈ X.
- Define an objective set Xg ∈ X.

H. Configuration Space

The configuration space includes all possible configurations that the RC can take in space. In the case presented in this paper, the space is considered to be 2D (The RC always moves in the ground plane, z = 0), however the RC configuration has to be defined by three parameters, since it is a rigid body with relevant orientation. Thus, each configuration is defined by two position values (x, y) and one orientation value (θ). Within the set of configurations C, two relevant complementary subgroups are defined, C_{free} and C_{obs}.

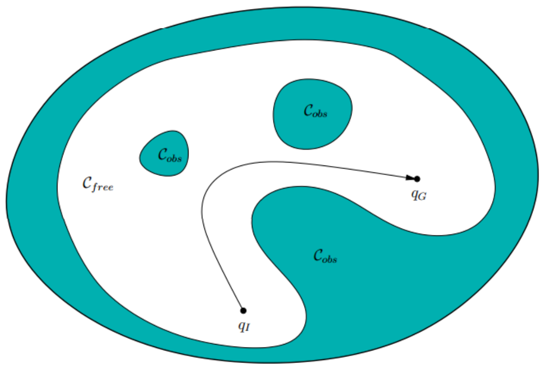

Fig. 8 – Configuration space example

C_{free} is called free space and consists of configurations that avoid obstacles, while C_{obs} is just the opposite, the set of configurations that match the obstacles.

I. Grid-Based Search

The principle behind this type of algorithm is to define a grid above the space in which it is intended for the system to navigate, and each position of the grid (i,j) corresponds to a possible configuration. Within the grid, the RC can move to any point adjacent to the position in which it is.

The definition of grid size results from a trade-off between computation time and motion precision, i.e. the larger the unit of the grid, the faster the path computation will be, however, a Grid with very large divisions may not admit paths between obstacles because they are in adjacent grid positions, whereas in reality there is sufficient space to pass between them. This problem can result in paths that deviate from the optimum path or the non-existence of a solution.

J. A Algorithm*

The A* algorithm [17] [18] follows a search model of paths called forward search, like this,

```
FORWARD_SEARCH
Q.Insert($x_I$) and mark $x_I$ as visited
while Q not empty do
    $x \leftarrow$ Q.GetFirst()
    if $x \in X_G$
        return SUCCESS
    forall $u \in U(x)$
        $x' \leftarrow f(x, u)$
        if $x'$ not visited
            Mark $x'$ as visited
            Q.Insert($x'$)
        else
            Resolve duplicate $x'$
return FAILURE
```

At every moment of the search for paths, there are three types of states:

- **Unvisited** - States that, as the name implies, have not yet been tested by the algorithm. All points are points not initially visited, except for the starting point x_i.
- **Closed** - States already visited and for which, already all possible consequent states have been tested. A consequential state x is a state x 'for which there exists an action u ∈ U (x) such that x' = f (x, u).
- **Open** - States already visited, but that may still have unaccompanied states. Initially the only living state is the initial state x_i.

This algorithm is an extension of the Dijkstra's algorithm [19] that attempts to reduce the number of states explored by introducing a heuristic estimation of the cost to reach the target state. Its formulation is done by creating possible path trees with an associated cost, starting from a specific point and continuing each one of these paths until it reaches the desired objective point.

Let C (x) be the cost of going from x_i to x, and G (x) denote the cost associated with going from x to a state x_g. C*(x) should be possible to calculate by dynamic programming, however, it is not possible to know in advance the value of G*(x). What happens is that in many applications, it is possible to reasonably underestimate this cost. An example of such an estimate would be to consider the straight-line distance between a starting point and an end point, since this cost estimate ignores possible obstacles. When obstacles are introduced, this cost can only increase. The objective will be to have an estimate as close as possible to the optimal cost, without exceeding that same cost. Let G* (x) be this estimate. In this algorithm, we use the sum C* (x) + G*(x), implying that the path search order is made from the estimated cost G*(x).

K. Global Algorithm

The implemented algorithm is composed by two different sections, one that is defined as the algorithm initialization and is only executed once, when the system is turned on, for the creation of the first map, and a second section where a loop cycle is executed until a termination order from the user.

After the initialization of the algorithm, the obstacle positions are calculated and the developed path planning method is applied. The route is defined by a set of commands that guide the RC between the positions of the map mesh.

The first command of the route is executed and the next command is evaluated. At this point, the system has two options. If the next command is in the same direction as the executed command, the map can be updated, then a new image is captured and then the evaluated command is executed, moving the system in a linear fashion. At this time, none of the commands in the previous route will be executed. After the movement, a new image is captured and the system returns to the obstacle mapping process, restarting the cycle.

The other option, when the command following the current command is evaluated, is this command having a direction different from the current direction, thus making it impossible to capture two linearly followed images. The system then continues on the previously calculated route, iterating the number of the command to execute and returning to the execution step of commands.

IV. System Identification

Due to bad IMU data measurements, the system identification ended up being done relying only on the position data through time. To avoid sliding of the RC tires, the power used for the tests was always constant and low.

L. Straight-line

Regarding the path in a straight-line, it was assumed that, given constant power, the position would change in relation to time following a third-degree polynomial curve until stationarity was achieved. This turned out to be a fair assumption as it will be shown.

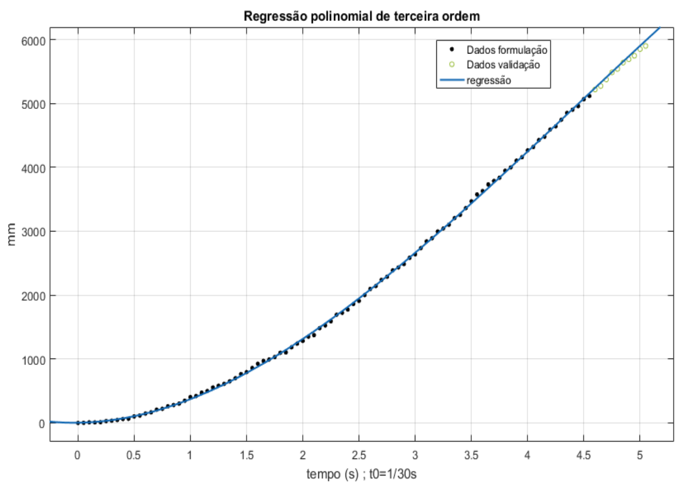

Fig. 9 – 3rd degree polynomial regression

The function found that describes the regression represented in the graph is as follows:

$$p(t) = -30.44t^3 + 385.6t^2 + 3.835t \tag{14}$$

The first derivative of p (t) determines the velocity function v(t) and the second derivative determines the acceleration function a(t) of the system.

$$v(t) = -91.32t^2 + 771.2t + 3.835 \tag{15}$$

$$a(t) = -182.64t + 771.2 \tag{16}$$

Knowing that in uniform linear motion the speed is constant and the acceleration is zero, the acceleration function is equal to zero in order to find the point at which this regime is reached.

$$0 = -182.64t + 771.2 \quad \rightarrow \quad t = 4.2s \tag{17}$$

Substituting t = 4.2s into the velocity equation determines the stationary terminal velocity.

$$\begin{gathered} v(4.2) = -91.32 \times 4.2^2 + 771.2 \times 4.2 + 3.835 \\ \rightarrow \; v_{final} = 1632\ mm/s \end{gathered} \tag{18}$$

Assuming that on the last second of measured points gathered the system is already at uniform linear motion, a linear regression was performed on that set of data which resulted in the function:

$$p(t) = 1611t - 5250\ mm \tag{19}$$

Deriving the position function previously stated, the stationarity velocity of the system, v(t) = 1611 mm/s, is found.

Comparing this final velocity value with the value found in the linear regression, it is noticed that the value from the polynomial function is a little higher and that on the final seconds of data, the polynomial regression is starting to deviate from the points measured, so, a final composed regression was drawn from the t = 4.2s point.

$$p(t) = \begin{cases} -30.44t^3 + 385.6t^2 + 3.835t\ , t \le 4.2 \\ 1611t - 5250\ , t > 4.2 \end{cases} \tag{20}$$

M. Curves

Tests were performed for the maximum direction values for the two directions, left and right, and what was important to model was the movement of the RC in the extension from 0 to 45º and 0 to -45º of orientation, since in the algorithm implemented, in no position the system takes different orientations from this range.

Since the points from the data measured didn't draw a perfect circle from the start position on, the data was approximated by ellipses.

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1 \tag{21}$$

For the left hand turn (21) describes the ellipse formed by the trajectory of the vehicle, where a and b are the values of the largest and smallest radius respectively of the ellipse, and C(h,k) is the center of the ellipse. All these values are constant:

a = 597, b = 562, C(h,k) = (-597,0)

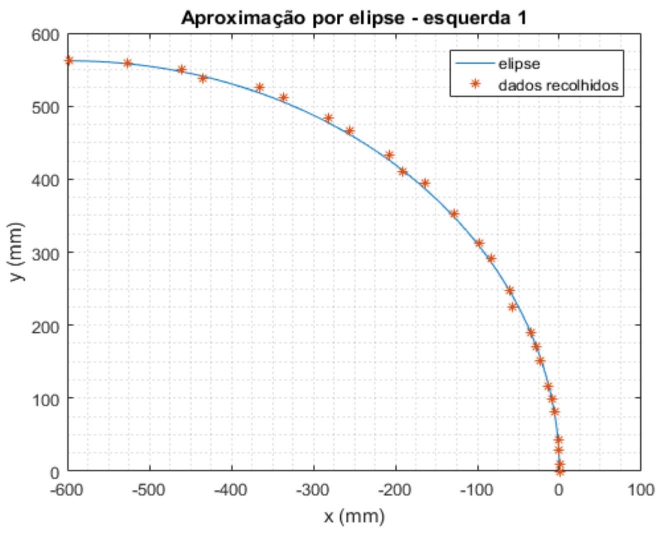

Fig. 10 - Ellipse approximation to left-hand turn

After having an expression for the ellipse, one has to know at which point the orientation of the RC is equal to -45º, to know in what position in space is the system and how long it took to get there. Thus, knowing that the orientation of the RC is equal to the angle formed by the slope of the tangent of the curve and the y-axis, it is necessary to calculate the tangents along the previous curve and find the point at which the tangent equals -45°.

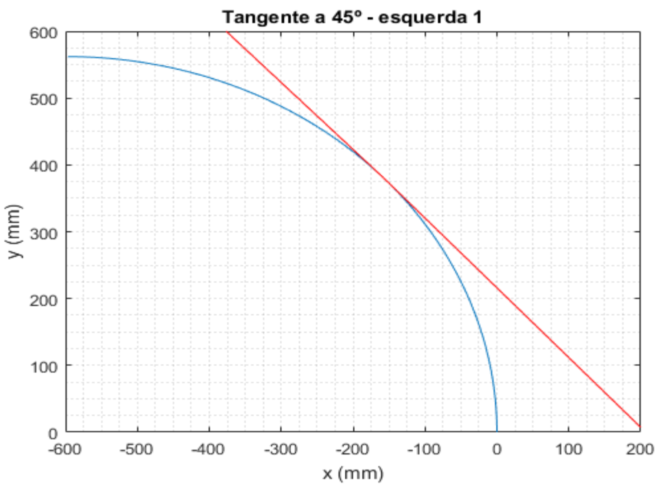

Fig. 11 - Tangent to the left-hand turn ellipse

In the graph from Fig. 11 it is drawn once again the curve made by the RC and also the tangent with slope equal to -45º that allows to find the position in space where the system is when it has this orientation is (-161.8,384.7) mm. Knowing the position occupied, it remains to know how long it takes to reach this position, and this is achieved by comparing the position coordinates with a coordinate curve and time taken in the test. As the values collected in the test are of a discrete nature, the calculated exact position is not included in any of the points collected, so that interpolation could be done between the nearest points, but since point-to-point differences are only half second, and that one of the collected points p = (-164,394) is very close to the desired point, the desired time is approximated to the time it took to arrive at this point, arriving at t(p) = 0.75s.

For the right-hand turn, the same approach was taken and the values gathered for the ellipse equation are:

$$a = 478,\ b = 511,\ C(h,k) = (478,0)$$

Equating the tangent of the curve and doing the same procedure as was done for the left-hand turn, the position in space that the system occupies at the point where the drawn tangent touches the ellipse is (155.4,377.1).

Again, the time taken for the system to reach the point at which its orientation was equal to 45 ° was ascertained. As before, due to the fact that the collected data is discrete, the exact point at which the time was intended did not correspond to a point in the data, so it was approached by the nearest point, which was at the position p = (157,368). The time the system took to arrive at this point was then t(p) = 0.85s.

V. Tests and Results

N. Parameters evaluation

The grid's size for the system's map was defined as a 3x3m divided by 10x10 squares of 30x30cm each.

To test the accuracy with which the RC executes the paths delineated by the path planning algorithm, three tests were performed, where paths were inserted manually and the differences between expected positions and effective positions were evaluated through the length of the paths.

Table 1

	Mean error	
	x (mm)	y (mm)
Right-Left turn	-10,7	140,0
Left-Right turn	1,3	39,4
Straight-line	-10,8	-20,2

From Table 1 it can be concluded that the largest position errors occur in the y coordinate, and mainly after a right-left turn, although this was somewhat expected since the variations in the y coordinate occur every step of the way, while in the x coordinate there are only changes when there are turns, so it's supposed to accumulate less error than the y coordinate.

O. Results of system tests

Thirty tests were performed, where several different environments were disposed in front of the system, so that every aspect of the implementation could be evaluated.

Table 2

No. of identified clusters	**No. of occurrences**	**Obstacle position**	**Path Planning**	**Total**
1	18	1,2	0,09	1,29
2	31	2,3	0,11	2,41
3	29	3,3	0,1	3,4
4	13	4,4	0,12	4,52
5	7	5,4	0,13	5,53

Adding the values from the second column of the previous table leads to the total number of times the map was updated in the thirty tests performed (98). By dividing this value by the total number of tests, an average ratio of map updates per test equal to 3.27 is reached, which amounts to 0.93 map updates per meter.

By evaluating the remaining columns of the table, it is possible to conclude that the number of clusters identified by the process, as expected, only affects the time spent calculating obstacle positions, increasing with the growth in the number of clusters identified, without affecting the processing time of the path planning algorithm.

In the 98 map updates verified in the tests, problems occurred multiple times that led the path planning algorithm to calculate a path that collided with the obstacles in the environment, however, many times errors in some parts of the layout of the map that led to the system being on a collision course were dissipated by a new map update and correction of what had been previously calculated. These problems derive from not completely linear system movements or outliers in the clustering method.

Table 3

	Wrong obstacle position	**Unidentified obstacle**	**Inexistent obstacle identified**
No. of ocurrences	15	7	4
Frequency per test (%)	50,0	23,3	13,3
Frequency per map (%)	21,7	10,1	5,8

From table 3, the frequency with which an obstacle is not identified or that, on the other hand, a non-existent obstacle is identified does not seem high, but with no comparative reference, nothing can be concluded. Unlike the two columns on the right, the values in the first column appear to be high and prohibitive for the system to work, however, these values can be somewhat misleading since it was considered that a wrong position of an obstacle will be a deviation in any direction from a position in the map mesh in relation to the expected position.

Table 4

	Paths leading to colision	**No. of colisions**
No. of ocurrences	7	2
Frequency per test (%)	23,3	6,7
Frequency per map (%)	10,1	2,9

The problems listed in table 13, by themselves, do not elucidate the effects of misclassification of obstacles. Table 14 shows the results of this defective mapping. First of all, it is important to check, by relating the two previous tables, that of the total of 26 defects verified in the mapping, only 7 calculated routes resulted which would result in a collision with some of the obstacles, which corresponds to approximately one route in a collision course with every ten updates of map. Due to the map updates, only two collisions occurred during the 30 tests performed, however the goal was not to collide, at all.

VI. Conclusions and Future work

P. Conclusions

The main idea behind the calculation of distances to obstacles of unknown shapes and sizes in space is based on the premise that these obstacles do not change these characteristics over time, and as such it is possible to obtain relations between the dimensions of obstacles by capturing images after linear movement of the camera, even without prior knowledge of the characteristics of the objects. These differences in dimensions are sufficient, together with the focal length value of the camera to calculate distances from the camera to the obstacles. Having said this, it was discussed how to obtain information about the objects through the camera. In order to extract information from an image and compare this information with another image in sequence, we used methods of detection of characteristics such as the SIFT and SURF methods, and the SURF method was implemented in the system algorithm because it presented similar results to the SIFT method and was a faster processing method.

One of the crucial points, and also more difficult to implement with validity, for the system to work is the segmentation of obstacles in the captured images. For this part of the algorithm, instead of segmenting each image individually, using the perception that objects closer to the camera have larger displacements and, more distant objects, smaller displacements between two images, vectors are formed that join the points of interest paired in the images, segmenting the vectors. After the vectors were formed, a clustering method (DBSCAN) was used, which would then segment, without having to input a defined number of groups as input. In addition, this method is important, since it is robust to the appearance of outliers, thus eliminating vectors formed by wrongly matched points.

After calculating the distances to the obstacles by the mentioned methods, it was necessary to resort to a method that decided which direction to take each moment. For this we used a route planning algorithm that took into account the mapping of the obstacles encountered. The algorithm A * was chosen because it is a relatively fast algorithm in computational terms and versatile, since it is based on a principle of grid-based search, where it is possible to define the size of the grid to be disposed on the Map and the start and end points. This method creates a path between the starting point and the target point, avoiding the obstacles on the map. Since it was intended that the system function autonomously, the paths calculated by the algorithm would have to be updated as the system moved. Thus, it was defined that as the system progressed, the camera would capture two images at different positions on the map after linear motion, and the algorithm created a new map with the obstacles displayed, then processing a new way to go, restarting the cycle.

Before performing the tests to the operation of the system, tests were made to identify the parameters of movement of the system. To move on the grid defined on the map, the system could only take three different orientations in each position and would have at most three different positions to move next, so it was necessary to notice the movement of the RC in space. After this process, the movement in the defined grid had to be adapted, since the movements of the RC did not allow to reach the positions and orientations, previously defined, of the way the map was defined.

Finally, the system was tested. It was initially verified the validity of the parameters resulting from the system identification, where it was noticed that there were some discrepancies between the expected positions and the effective positions, and that these could influence the calculations of the positions of the obstacles in the global system. To test the system, 30 tests were performed, in which the system was exposed to several different scenarios. From these tests it was found that in two of them the system collided with some of the obstacles in its path due to poor identification of the environment, which resulted in a definite path leading to collisions.

The objective of the work was to get the system to move autonomously avoiding collisions. Although a small number of collisions have occurred, it has been found that in 2.9% of the times that the map is updated and a new route planned, the system ends up colliding with an obstacle. Thus, it is concluded that the system is not effective in avoiding collisions, and as such the objective has not been fully achieved. However, the method introduced for the calculation and positioning of obstacles in space is effective, although computationally heavy, as well as the method of route planning, and the major problems for the correct functioning of the system as a whole relate to the clustering process and the small differences between expected positions and effective positions of the system.

Q. Future Work

Taking as a starting point the work done here, and realizing from the last paragraph of the conclusions that the problems found in the operation of the system concentrate mainly on the control of movement of the system and in the process of segmentation, the following follow-up points are proposed for works Improvement of this thesis:

- Study of new approaches to individual image segmentation, identifying only points of interest in the image for calculations of distances to obstacles;
- Possibility of introducing a camera that allows the variation of the focal distance to be able to have depth perception, facilitating the process;
- Study of new processes that allow to extract information of sequence of images, for example registration of images;
- Introduction of control for the position of the RC in the space so that the movements are more precise, improving the perception of the system of the environment;

References

[1] J. Lutin, A. Kornhauser and E. Lerner-Lam, "The Revolutionary Development of Self-Driving Vehicles and Implications for the Transportation Engineering Profession," *Institute of Transportation Engineers. ITE Journal; Washington,* pp. 28-32, 2013.

[2] M. Bertozzi, A. Broggi and A. Fascioli, "Vision-based intelligent vehicles: State of the art and perspectives," *Robotics and Autonomous Systems,* vol. 32, pp. 1-16, 2000.

[3] N. Martins, "Integration of RC Vehicles in a Robotic Arena," MSc Thesis, Mestrado Integrado em Engenharia Aeroespacial, Lisboa, Jan 2017.

[4] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

[5] A. Saxena, M. Sun and A. Ng, "Make3D: Depth Perception from a Single Still Image," Chicago, USA, In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, July 13–17, 2008, pp. 1571-1576.

[6] P. Alizadeh, "Object Distance Measurement Using a Single Camera for Robotic Applications," Ontario, Canada, 2015.

[7] R. Hartley and A. Zisserman, in *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2004, pp. 237-279.

[8] J. Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.

[9] S. Yang and Y. Hu, "Robot Path Planning in Unstructured Environments Using a Knowledge-Based Genetic Algorithm," School of Engineering, University of Guelph, Canada, 2005.

[10] S. M. LaValle, Planning Algorithms, Cambridge: Cambridge University Press, 2006, pp. 185-248.

[11] S. M. LaValle, Planning algorithms, Cambridge: Cambridge University Press, 2006, pp. 27-74.

[12] "wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN. [Accessed 16 Janeiro 2017].

[13] C. Harris and M. Stephens, A combined corner and edge detector, United Kingdom: The Plessey Company plc., 1988.

[14] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," University of British Columbia, Vancouver, B.C., Canada, 2004.

[15] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," Katholike Universiteit Leuven, Zurich.

[16] "wikipedia\DBSCAN," [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN. [Accessed 13 Fevereiro 2017].

[17] J. Crowley and R. Stern, "Fast computation of the difference of low pass transform," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1984, pp. 212-222.

[18] P. E. Hart, N. J. Nilsson and B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics, pp. 100-107.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, MIT Press and McGraw–Hill, 2001, p. Section 24.3: Dijkstra's algorithm.

[20] J. Ponce and D. Forsyth, in *Computer Vision A modern approach*, pp. 321-345.

[21] "Scholarpedia\SIFT," [Online]. Available: http://www.scholarpedia.org/article/SIFT. [Accessed 7 Nov 2016].