

Article

3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms

Wilbert G. Aguilar ^{1,2,3,*} and Stephanie G. Morales ^{2,*}

¹ Centro de Investigación Científica y Tecnológica del Ejército CICTE, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador

² Departamento de Eléctrica y Electrónica DEEE, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador

³ Research Group on Knowledge Engineering GREC, Universitat Politècnica de Catalunya UPC-BarcelonaTech, Barcelona 08040, Spain

* Correspondence: wgaguilar@espe.edu.ec (W.G.A.); sgmorales2@espe.edu.ec (S.G.M.)

Academic Editor: Mostafa Bassiouni

Received: 19 August 2016; Accepted: 26 September 2016; Published: 18 October 2016

Abstract: This paper describes a 3D path planning system that is able to provide a solution trajectory for the automatic control of a robot. The proposed system uses a point cloud obtained from the robot workspace, with a Kinect V2 sensor to identify the interest regions and the obstacles of the environment. Our proposal includes a collision-free path planner based on the Rapidly-exploring Random Trees variant (RRT*), for a safe and optimal navigation of robots in 3D spaces. Results on RGB-D segmentation and recognition, point cloud processing, and comparisons between different RRT* algorithms, are presented.

Keywords: path planning; RRT; RRT*; point cloud registration; 3D modeling; mobile robotics; RGB-D segmentation; computational geometry

1. Introduction

Robotic systems have been playing an important role in the modern world. They are applied in several areas: in manufacturing, entertainment, the toy industry, the medical field, exploration, military and multimedia applications [1–3]. Most commercial robots need manual programming or teleoperation for the executing of movements. This important task requires human intervention that results in long set-up times, and the expertise of operators to control robots. Autonomous navigation is an alternative for solving this issue. Flexibility and energy efficiency are also important achievements of modern robotics [4,5].

Navigation includes algorithms for perception and motion estimation, and also for path planning and optimization in order to connect the start point with the goal point. Several techniques, such as proximity sensors, have been used for perception. However, 3D point cloud processing is one of the most important areas of computer vision because it obtains a better description of the world. Different range imaging cameras are used to capture depth maps, such as stereo cameras, structured light scanners, time-of-flight cameras and others.

In this system, we use a low-cost 3D sensor to generate the 3D binary occupancy grid map and to describe the workspace. Our approach consists of a path planning system that obtains the environment description from a Kinect V2 point cloud, recognizes the start and goal point by computer vision techniques and solves the path planning issue using a variant of the RRT algorithm. The system includes calibration between the Kinect coordinate frame and the workspace coordinate frame using a registration technique.

Path planning determines a collision-free path between start and goal positions in a workspace clustered with obstacles [6]. Different path planning algorithms are used to find an optimal solution trajectory. Sample-based methods are widely used because of their effectiveness and low computational cost on high dimensional spaces. These methods use a representative configuration space and build a collision-free roadmap connecting points sampled from the obstacle free space.

The Probabilistic Roadmaps (PRM) and the RRT are sample-based methods. The RRT algorithm has been commonly used for a fast trajectory search because of its incremental nature. There are several versions of the RRT developed to improve the cost of the solution path. In our work, we compared the RRT with the RRT* algorithm and two other variations using the path cost as evaluation metrics.

This paper is organized as follows: The next section is focused on a brief description of related works. In Sections 3–5 we introduce the methods proposed to solve each part of the system in order to achieve the goal: registration, RGB segmentation and RRT path planning methods. Finally, experimental results and conclusions are presented in the last two sections.

2. Related Work

Research groups on robotics and computer vision have developed different techniques for robot navigation such as 3D perception and mapping based on RGB-D cameras [7], laser range scanners [8] and stereo cameras [9]. The Kinect was released in November 2010 as an RGB-D commercial camera, and after that, several applications on robotics appeared [10,11].

Segmentation and object recognition are fundamental algorithms for vision applications. Recently, the growing interest in 3D environments for robot navigation has increased the research on 3D segmentation. In [12], a color and depth segmentation is proposed to detect objects of an indoor environment by finding boundaries between regions based on the norm of the pixel intensity. A novel technique for depth segmentation uses the divergence of a 2D vector field to extract 3D object boundaries [13].

There are also several works on the literature for path planning to connect start and goal locations. The Ant Colony Algorithm, based on the behavior of ants searching for food, is used in many research projects due to its global optimization performance in multi goal applications [14]. Others use neural networks to give an effective path planning and obstacle avoidance solution that leads the robot to the target neuron that corresponds to the target position [15,16]. Some other strategies use behavior-based navigation and fuzzy logic approaches. In this solution, the image provides the data for the fuzzy logic rules that guide the robot to the safest and most traversable terrain region [17].

In this paper, we use a probabilistic path planning algorithm. Several optimizations have been developed in this area such as the RRT-Connect approach [18] that uses a simple greedy heuristic. Other variants have been presented in [19,20]. Our contribution is focused on the overall system of an optimal robot path planner, comparing the resulting paths of two different variants of the RRT* algorithm.

3. Workspace Data

A path planning algorithm requires a workspace, the initial and goal positions, and a robot. Our proposed system has three stages: environment perception, detection of target points and path planning.

In 2013, the Kinect V2 was introduced with a different technology with respect to the previous Kinect V1. The Kinect V2 is based on the measurement principle of time of flight: the distance to be measured is proportional to the time needed by the active illumination source for traveling from the emitter to the target [21]. Therefore, a matrix of time of flight cameras allows us to estimate the distance to an object, for each pixel. The Kinect V2 also has an RGB-D camera and an infrared camera.

In Table 1, Kinect V2 characteristics and technical specifications are presented.

Table 1. Kinect V2 sensor characteristics and specifications. Data from [22].

Characteristics	Specifications
RGB camera resolution	1920 × 1080 pixels
Depth camera resolution	512 × 424 pixels
Field of View ($h \times v$)	70 × 60 degrees
Depth operating range	0.5–4.5 m
Object pixel size	between 1.4 mm (@ 0.5 m range) and 12 mm (@ 4.5 m range)

Depending on the distance between the object point and sensor, the pixel size has an estimated value between 1.4 mm at 0.5 m of depth and 12 mm at 4.5 m of depth, as shown in Table 1, giving the approximate resolution of our point cloud.

We have used the open-source software developed by Hatledal, L.I. [23] to obtain the colored point cloud in millimeters from the delivered depth and RGB data. This point cloud will be processed with the proposed algorithm. We align the point cloud relative to the Kinect, to the point cloud relative to the workspace coordinate frame (real physical set of points), previous to image segmentation. A technique to solve this issue is point cloud registration. The objective of registration is the matching of two point sets by the estimation of the transformation matrix that maps one point set to the other [24,25]. A widely used registration method is the Iterative Closest Point (ICP) due to its simplicity and low computational complexity. ICP assigns correspondence based on the nearest distance criteria and the least-squares rigid transformation giving the best alignment of the two point sets [26].

Assuming that the point set $\{m_i\}$ and $\{d_i\}$, $i = 1 \dots N$ are matched and related by:

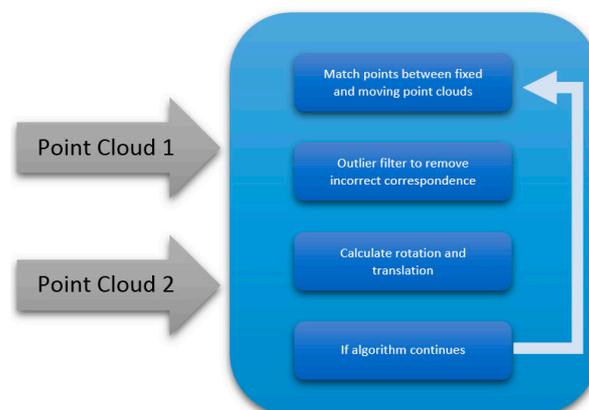
$$d_i = Rm_i + T + V_i \quad (1)$$

where R is a 3×3 rotation matrix, T is a 3D translation vector and V_i is a noise vector [27].

The goal is to obtain the optimal transformation for mapping the set $\{m_i\}$ onto $\{d_i\}$ minimizing the least-square error:

$$\Sigma^2 = \sum_{i=1}^N \|d_i - \hat{R}m_i - \hat{T}\|^2 \quad (2)$$

The least-squares solution is not optimal if there are many outliers or incorrect correspondences on the dataset, and therefore other techniques should be added. Several algorithms are able to compute the closed-form solution for R and T . In this work, we use the ICP algorithm to register two point clouds, as presented in Figure 1.

**Figure 1.** Point cloud rigid registration algorithm [28].

This algorithm uses an outlier filter before the transformation matrix calculation, in order to solve the incorrect correspondence issue.

We find the rigid transformation matrix ($T_{4 \times 4}$) that aligns point set P to point set Q so:

$$Q = TP \tag{3}$$

where $P = [x, y, z, 1]^T$ is a vector with the centroid coordinates of the boundary circles extracted from the point cloud in the work plane, and $Q = [x', y', z', 1]^T$ are the corresponding data points from the workspace coordinate frame, as shown in Figure 2.

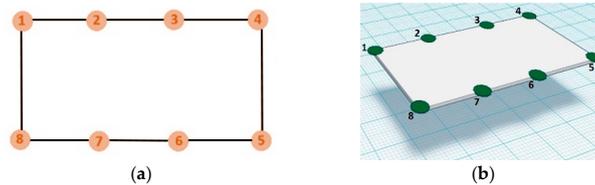


Figure 2. Registration point sets. (a) In orange the feature points extracted from the image; (b) In green the corresponding workspace coordinate frame points.

With this technique, we get the point cloud on the robot coordinate frame suitable for feature extraction and path planning algorithms.

4. Target Points

Image segmentation is the process of separating an image into regions. Each region is defined as a homogenous group of connected pixels with respect to a selected interest property, such as color intensity or texture [29]. For this purpose, one of the most effective approaches is segmentation using color threshold [30]. This method is used in object recognition, assuming that pixels with a color value, inside of a defined range in the color channels, belong to the same class or object. A threshold image can be defined by:

$$g(u, v) = \begin{cases} 1, & f(u, v) > t \\ 0, & f(u, v) \leq t \end{cases} \tag{4}$$

where $f(u, v)$ is a set of pixels from the input image, $g(u, v)$ is the output binary image (binarization) and t is the threshold value [31] (a threshold value is needed for each color channel).

In our work, the input image is generated from the point cloud, so each pixel has a corresponding xyz coordinate. The thresholds are obtained locally by selecting a target region pixel to obtain its color component and a sensibility value chosen experimentally. The threshold of our image was defined by:

$$g(u, v) = \begin{cases} 1, & p_R - s \leq f(u, v) < p_R + s \\ & p_G - s \leq f(u, v) < p_G + s \\ & p_B - s \leq f(u, v) < p_B + s \\ 0, & otherwise \end{cases} \tag{5}$$

where p_R, p_G, p_B are the manually-selected pixel color components and s is the sensibility.

After the binary mask is obtained, we use morphological operations to remove noise and unwanted holes. Some of the main morphological operations that can be used are dilatation, erosion, opening and close.

Dilatation is applied to each element of the mask image A using a structuring element B , resulting in a new mask image with a different size. The dilatation $A \oplus B$ removes holes and indentations smaller or equal to the structuring element [32].

Erosion $A \ominus B$ removes noise smaller than the structuring element.

The opening $A \circ B$ eliminates edges and removes some of the foreground bright pixels.

The closing operation $A \cdot B$ keeps the background regions with a similar size of the structuring element, and removes all other pixels of the background.

We use the opening and close operations to enhance our binary image and obtain a better approximation of the interest regions.

The next step is representation where adjacent pixels of the same class are connected to form spatial sets s_1, \dots, s_m . For the representation of our binary image, we use the blob detection method of Corke P. [33]. Blob is a spatial contiguous region of pixels. Properties or features such as area, perimeter, circularity, moments or centroid coordinates are widely used to identify these regions.

We identify our interest regions by filtering the blobs based on their position and area in order to simplify the problem. Knowing the boundaries of the workspace and the minimum and maximum size of the target circles, we obtained the interest blobs and their centroid coordinates.

In Figure 3, we can see the segmentation process implemented in our system.

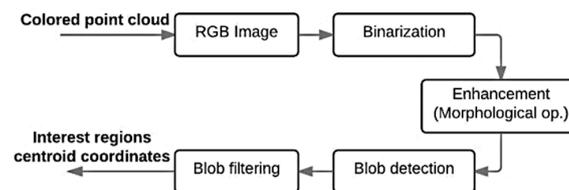


Figure 3. Segmentation process.

5. Path Planning

5.1. Travelling Salesman Problem

This path planning problem, known as the travelling salesman problem (TSP), is focused on the organization of points and the search of an optimal path through all of them. We have a set $\{c_1, c_2, \dots, c_N\}$ of cities and a Euclidean distance $d\{c_i, c_j\}$ for each pair $\{c_i, c_j\}$ of cities. The goal of the algorithm is to order the cities so as to minimize the total length of the tour [34].

Our solution for the TSP is based on the Nearest Neighbor (NN) algorithm shown on Algorithm 1. The performance of this approach is not the best in the literature, but NN gives suitable solutions with respect to our workspace and the possible locations of the target points. An example of the workspace and target points is presented in Figure 4.

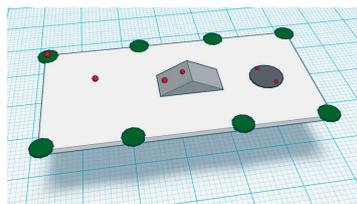


Figure 4. Target points in the workspace are shown in red, while the workspace boundary circles are shown in green.

The algorithm is approached as follows:

Algorithm 1: The nearest neighbor algorithm

- 1: $P = [p_1 p_2 p_3 \dots p_n]$
 - 2: $p_1 \leftarrow$ starting point
 - 3: $V \leftarrow p_{\text{nearest}}$ from unvisited points
 - 4: mark V as visited
 - 5: if all P are visited, end
 - 6: return to step 3
-

Once the points are ordered, we develop the path planner based on the RRT algorithm, as mentioned above.

5.2. RRT

Using the notation from [35], the path planning problem can be denoted as:

- X is a bounded connected open subset of \mathbb{R}^d where $d \in \mathbb{N}$, $d \geq 2$.
- $G = (V, E)$ is a graph composed by a set of vertexes V and edges E .
- X_{obs} and X_{goal} , subsets of X , are the obstacle region and the goal region respectively.
- The obstacle free space $X \setminus X_{\text{obs}}$ is denoted as X_{free} and the initial state x_{init} is an element of X_{free} .
- A path in X_{free} should be a collision-free path.

In path planning algorithms, there are two variations: the feasibility problem that refers to the searching of a feasible path and the failure report otherwise; and the optimization problem that focuses on a feasible path with minimal cost. The cost is $c : \sum X_{\text{free}} \rightarrow R > 0$ assigned to a non-negative cost of each collision-free path. A feasible path is a collision free-path that starts in X_{init} and ends in X_{goal} .

In this part, we introduce the standard RRT algorithm and the RRT* variations tested in our system. These algorithms rely on the following functions:

- Function *Sample*: returns x_{rand} , an independent identically distributed sample from X_{free} .
- Function *Nearest Neighbor*: returns the closest vertex x_{near} to the point $x \in X_{\text{free}}$, in terms of the Euclidean distance function.
- Function *Steer* returns a point x_{new} at a distance ε , from x_{near} in direction to x_{rand} .
- Function *Obstacle Free*: Given two points $x, x' \in X_{\text{free}}$, the function returns true if the line segment between x and x' lies in X_{free} and returns false otherwise.
- Function *Near Vertices*: returns a set V' of vertices that are close to the point $x \in X_{\text{free}}$ within the closed ball of radius r centered at x .
- Function *Parent*: returns the parent vertex $x_{\text{parent}} \in E$ of $x \in X_{\text{free}}$.

As shown in Figure 5, the RRT algorithm starts in the initial state. For each iteration, the graph incrementally grows by sampling a random state x_{rand} from X_{free} , and connecting x_{new} returned by function *Steer* with the nearest vertex in the tree x_{nearest} , when the connection is obstacle-free. x_{new} is added to the vertex set and $(x_{\text{nearest}}, x_{\text{new}})$ to the edge set. This algorithm is detailed in Algorithms 2 and 3.

The connections between two edges, considered in this paper as straight-line connections, are made by increasing the distance from the start position using the local planner.

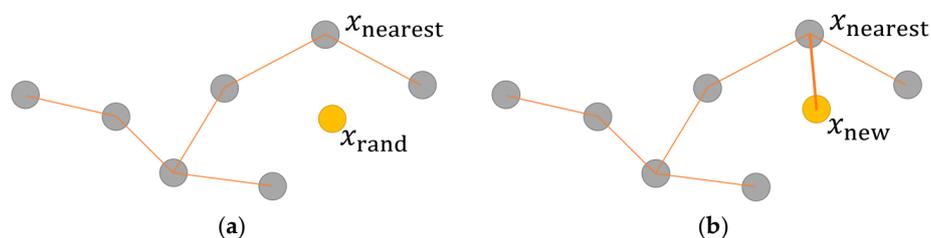


Figure 5. Growth of the tree. In gray the current tree vertices, in yellow the new vertex. (a) x_{rand} returned from sample function; (b) extending tree from the nearest point at ε distance on x_{free} on x_{rand} direction.

Algorithm 2: RRT main algorithm

```

1:  $V \leftarrow \{x_{\text{init}}\}; E = \emptyset; i = 0;$ 
2: while  $i < N$  do
3:    $G \leftarrow (V, E);$ 
4:    $x_{\text{rand}} \leftarrow \text{Sample}(i);$ 
5:    $i \leftarrow i + 1;$ 
6:    $(V, E) \leftarrow \text{Extend}(G, x_{\text{rand}})$ 

```

Algorithm 3: Extend function of the RRT algorithm

```

1:  $V' \leftarrow V; E = E';$ 
2:  $x_{\text{nearest}} \leftarrow \text{Nearest}(G, x);$ 
3:  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x);$ 
4: if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
5:    $V' = V' \cup \{x_{\text{new}}\};$ 
6:    $E' = E' \cup \{(x_{\text{nearest}}, x_{\text{new}})\};$ 
7: return  $G' = (V', E')$ 

```

5.2.1. RRT* Algorithm

The RRT* algorithm adds vertices in the same way as the RRT (Algorithm 2), but has two optimization procedures in the extend function shown in Algorithm 4. These optimizations connect vertices to obtain a path with minimum cost. The first one, done after x_{new} , is added to the vertex set of the tree. The function Near Vertices returns the set of vertices X_{near} . If the obstacle-free connection between x_{new} and one of the vertices of X_{near} has a lower cost than the path through x_{nearest} then $(x_{\text{near}}, x_{\text{new}})$ is added to the edge set, and the other connections are ignored. An example of this optimization is shown in Figure 6.

Algorithm 4: Extended function of the RRT* algorithm

```

1:  $V' \leftarrow V; E = E';$ 
2:  $x_{\text{nearest}} \leftarrow \text{Nearest}(G, x);$ 
3:  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x);$ 
4: if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
5:    $V' = V' \cup \{x_{\text{new}}\};$ 
6:    $x_{\text{min}} \leftarrow x_{\text{nearest}};$ 
7:    $X_{\text{near}} \leftarrow \text{NearVertices}(G, x_{\text{new}});$ 
8:   for all  $x_{\text{near}} \in X_{\text{near}}$  do
9:     if  $\text{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$  then
10:       $c' \leftarrow \text{Cost}(x_{\text{near}}) + \text{Cost}(\text{Line}(x_{\text{near}}, x_{\text{new}}));$ 
11:      if  $c' < \text{Cost}(x_{\text{nearest}}) + \text{Cost}(\text{Line}(x_{\text{nearest}}, x_{\text{new}}))$  then
12:         $x_{\text{min}} \leftarrow x_{\text{near}};$  //Choose new parent for  $x_{\text{new}}$ 
13:         $E' = E' \cup \{(x_{\text{min}}, x_{\text{new}})\};$ 
14:     for all  $x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$  do
15:       if  $\text{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$  then
16:          $c' \leftarrow \text{Cost}(x_{\text{near}}) + \text{Cost}(\text{Line}(x_{\text{near}}, x_{\text{new}}));$ 
17:         if  $c' < \text{Cost}(x_{\text{near}})$  then
18:            $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$  //Rewire
19:            $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\};$ 
20:            $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\};$ 
21: return  $G' = (V', E')$ 

```

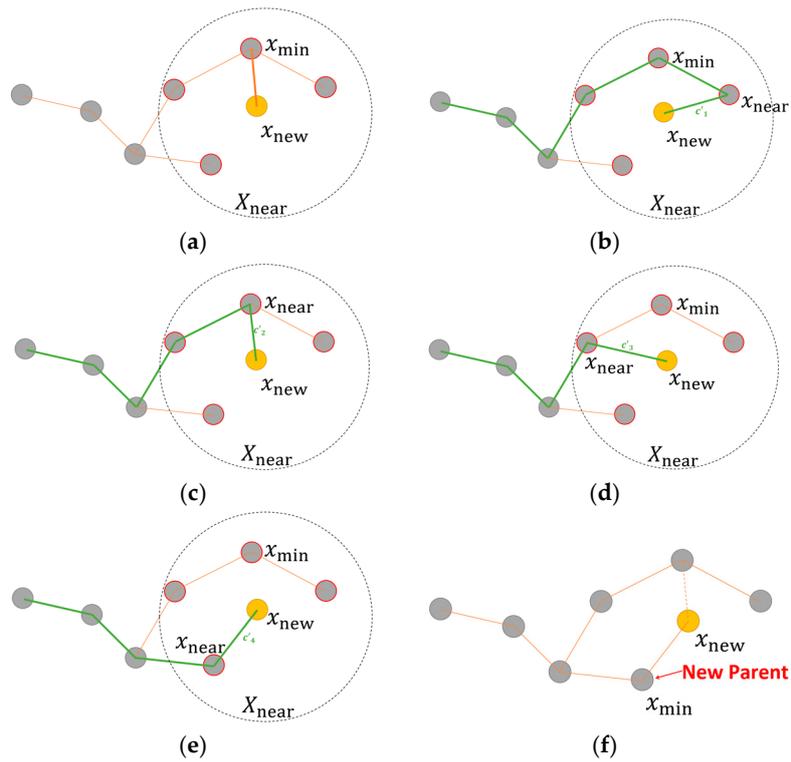


Figure 6. First optimization procedure of the Rapidly-exploring Random Trees variant (RRT*) algorithm. In gray the current tree vertices, in yellow the new vertex. (a) Near vertices to x_{new} ; (b–e) In green, the path from x_{init} to x_{new} through each x_{near} vertex; (f) new parent is chosen.

The second optimization procedure checks if the cost of one of the remaining vertices of X_{near} is lower when it is connected through the new vertex x_{new} . If it is lower, then the edge $(Parent(x_{near}), x_{near})$ is deleted and the edge (x_{new}, x_{near}) is added to the edge set E , as shown in Figure 7.

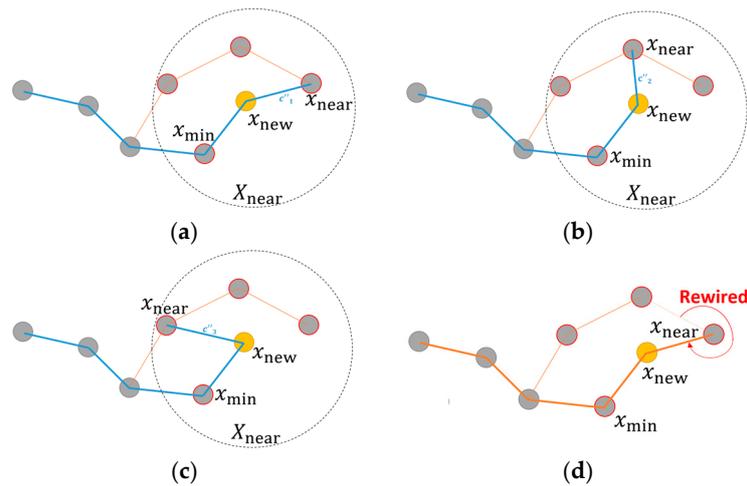


Figure 7. Second optimization procedure of the RRT* algorithm. In gray the current tree vertices, in yellow the new vertex. (a–c) In blue, the path from x_{init} to each vertex of x_{near} through x_{new} ; (d) If the cost function is lower than the cost of x_{near} through the current parent, a rewired procedure is executed.

The result is an optimal path where each vertex has the lowest cost. The result of the example is presented in Figure 8.

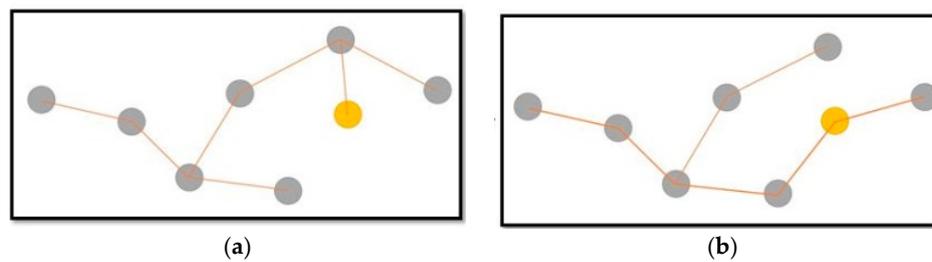


Figure 8. In gray the current tree vertices, in yellow the new vertex. (a) RRT standard resulting path; (b) RRT* optimal resulting path.

In order to decrease the time of finding a feasible solution, and to accelerate the rate of convergence and optimization, we proved two variations of the RRT*. The modifications were made on the Sample function by changing the probability of the generated random node. We have called these versions RRT* Goal, RRT* Limits and the combination of both RRT* Goal Limit (RRT* GL).

5.2.2. RRT* Goal Algorithm

In Algorithm 5, we present the sample function algorithm. In this function we guide the exploration of the tree in the direction of the goal region. We give a probability of 50% for sampling the goal region. The other 50% returns an identically distributed sample from X_{free} .

Algorithm 5: Sample (i , feasiblePath, X_{free}) function of the RRT* Goal algorithm

- 1: if ($i \text{ MOD } 2 = 0$) OR (feasible Path is TRUE)
 - 2: $x_{\text{rand}} \leftarrow \text{random}(X_{\text{free}})$;
 - 3: else
 - 4: $x_{\text{rand}} \leftarrow X_{\text{goal}}$;
 - 5: return x_{rand}
-

5.2.3. RRT* Limits Algorithm

Once the feasible path is found, we increase the probability of the random sample in a specific space. This accelerates the rate of convergence to an optimal, low-cost path.

As shown in Figure 9, we obtain the uniform distributed random sample by delimiting the range to the minimum and maximum coordinates (x, y, z) of the path found. This increases the density of random samples at the path surroundings, avoiding unnecessary optimizations in other sections of the tree. This is shown on Algorithm 6.

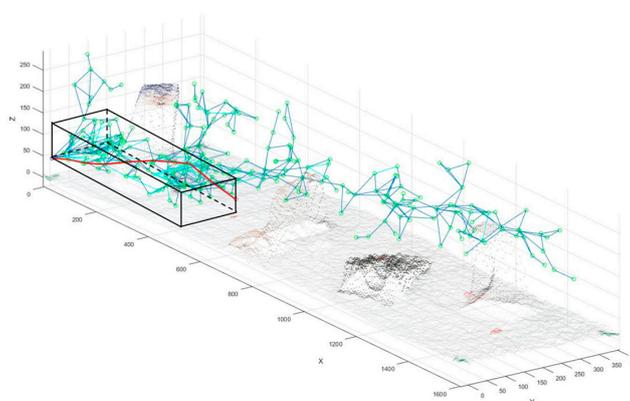


Figure 9. The edges of the tree are presented in blue. The feasible path is shown in red. The cube in black shows the limited space, for the random sample, generated on the RRT* Limits.

Algorithm 6: Sample (i , feasiblePath, V , X_{free}) function of the RRT* Limits algorithm

- 1: if feasible Path is TRUE
 2. $\min L = \min (V)$
 3. $\max L = \max (V)$
 4. $x_{\text{rand}} \leftarrow \text{random} (X_{\text{free}}, [\min L, \max L])$;
 5. else
 6. $x_{\text{rand}} \leftarrow \text{random} (X_{\text{free}})$;
 7. return x_{rand}
-

6. Experiment

The system was implemented on a laptop with the following characteristics: Processor Intel Core i7-4500 @1.90 GHz and 8.00 GB RAM. The graphics card used was a GeForce GT 735 (NVIDIA, Santa Clara, CA, USA).

A real 3D image captured by a Kinect V2 was processed off-line for three different scenes on the workspace. Each scenario had one, two and three obstacles, respectively, between the start and goal point.

The experiment was divided into three parts: (a) the alignment of the workspace point cloud of the Kinect to the workspace coordinate frame; (b) the target point recognition; and (c) the path planning algorithm with the variations. During the testing, we used a Cartesian robot to reach the target points and measure the obtained coordinates that do not depend on the robot dynamics [36–38].

6.1. Point Cloud Alignment

As mentioned above, the method used for point cloud alignment was the registration of two data sets. We rotated the Kinect for testing the transformation matrix effect on the original point cloud, as shown in Figure 10.

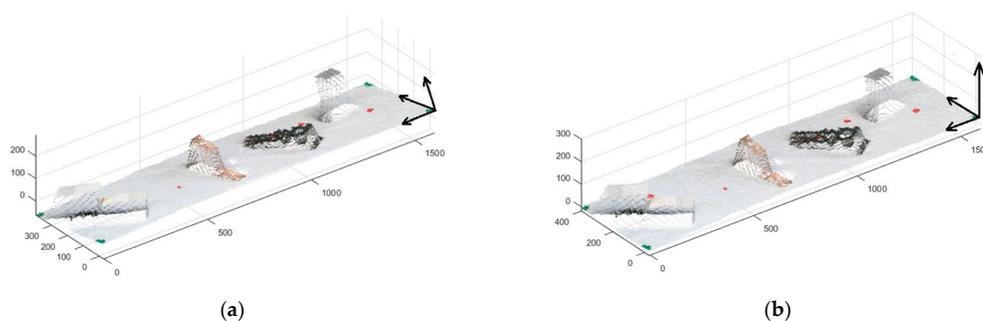


Figure 10. The black arrows represent the point clouds axis on the origin. (a) Point cloud on the Kinect coordinate frame; (b) point cloud on the robot coordinate frame.

In Table 2, we present information on eight workspace boundary circles (landmarks), their centroid coordinates estimated from the Kinect coordinate frame, the real coordinates on the workspace, the calibrated coordinates after registration and the error percentage of each circle centroid coordinate.

Because all of the circles are located on the same plane, we can calculate the average error percentage as the error of the workspace plane on the robot coordinate frame related to the Kinect pose by:

$$E_x \% = \frac{\text{calibrated}_x - \text{real}_x}{\text{delta}_x} \times 100 \quad (6)$$

where delta_x is the difference between the maximum and minimum value of the workspace seen by the Kinect on the x coordinate.

We can see in Table 2 that our system obtains an aligned point cloud with high accuracy.

Table 2. Results table of the point cloud alignment.

Landmark	Coord.	Kinect Coord. (mm)	Real Coord. (mm)	Calibrated Coord. (mm)	Error %
1	<i>x</i>	0.7	−1.5	−0.91	0.04
	<i>y</i>	79.9	0.1	−0.52	−0.16
	<i>z</i>	−125.7	2.3	−4.14	−6.44
2	<i>x</i>	9.1	3.85	−0.55	−0.27
	<i>y</i>	473.2	400.1	390.9	−2.19
	<i>z</i>	−158.7	−5.24	−2.43	2.81
3	<i>x</i>	351.7	347.46	345.7	−0.11
	<i>y</i>	71.6	−2.82	−7.73	−1.17
	<i>z</i>	−104.6	15.78	16	0.22
4	<i>x</i>	604.8	602.41	600.2	−0.14
	<i>y</i>	467.8	394.37	390.1	−1.02
	<i>z</i>	−134.4	16.58	20.11	3.53
5	<i>x</i>	942.5	942.56	940.9	−0.10
	<i>y</i>	65.1	−9.72	−14.4	−1.11
	<i>z</i>	−88.8	30.25	31.97	1.72
6	<i>x</i>	1308.3	1298.8	1297	−0.12
	<i>y</i>	453.6	386.97	377.5	−2.25
	<i>z</i>	−136.1	20.9	17.5	−3.40
7	<i>x</i>	1595.5	1591.8	1598	0.39
	<i>y</i>	50.9	−16.5	−25.26	−2.09
	<i>z</i>	−85.1	34.54	37.01	2.47
8	<i>x</i>	1603.4	1598.5	1596	−0.15
	<i>y</i>	452.5	383.1	374.5	−2.05
	<i>z</i>	−133.5	21.54	21.08	−0.46
–	–	–	–	Average	−0.2710

6.2. Target Points Recognition

In this section, in Table 3 we present the location on millimeters of five circle centroids recognized with the algorithm on the *xy* plane and the corresponding error percentage.

Table 3. Results table of the target points recognition.

Values	p1		p2		p3		p4		p5	
	<i>x</i>	<i>y</i>								
real	252.2	300.1	497.9	151.2	988.52	208.37	1194.3	300.1	1194.3	300.1
calculated	249.6	299	497	152.8	985.3	205.7	1187.8	299.2	1187.8	299.2
Error %	0.16	0.26	0.06	0.38	0.20	0.64	0.41	0.21	0.07	0.05

The error was also calculated with respect to the robot workspace seen by the Kinect. The points locations were obtained very accurately.

6.3. Path Planning

Each algorithm was tested ten times to obtain the mean cost, number of segments in the resulting path, the time for locating the target point and the execution time of the algorithm through all of the iterations.

We tested each algorithm with 300, 600 and 1000 iterations in order to compare their results with respect to path optimization.

These results are shown in Tables 4–12:

Table 4. Results. One obstacle, 300 iterations ^a.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	225	1501.85	31	4.94	4.94
RRT*	300	1469.5	26	4.43	15.81
RRT* Goal	300	1333.7	26	0.91	13.61
RRT* Limits	300	1474.4	28	5.98	17.09
RRT* GL	300	1302.4	27	0.82	16.76

^a RRT, Rapidly-exploring Random Trees; RRT*, RRT variant; RRT* GL, RRT* Goal Limits.

Table 5. Results. One obstacle, 600 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	225	1501.85	31	4.94	4.94
RRT*	600	1431.8	25	5.55	61.57
RRT* Goal	600	1289.9	26	0.86	54.73
RRT* Limits	600	1455.2	27	17.7	60.73
RRT* GL	600	1308.7	27	0.86	66.99

Table 6. Results. One obstacle, 1000 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	225	1501.8	31	4.94	4.94
RRT*	1000	1403.4	23	8.3	180.25
RRT* Goal	1000	1312.3	26	0.81	161.24
RRT* Limits	1000	1366.3	26	28.54	191.2
RRT* GL	1000	1299.1	26	0.86	198.01

Table 7. Results. Two obstacles, 300 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	190.5	1853.65	38	4.015	4.015
RRT*	300	1576	29	5	14.68
RRT* Goal	300	1477.2	30	1.15	11.93
RRT* Limits	300	1603.6	32	9.56	13.88
RRT* GL	300	1402.4	29	2.05	8.535

Table 8. Results. Two obstacles, 600 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	190.5	1853.65	38	4.015	4.015
RRT*	600	1433.7	25	2.49	66.4
RRT* Goal	600	1366	28	1.47	46.27
RRT* Limits	600	1512	31	7.86	56.19
RRT* GL	600	1359.6	28	1.19	65.4

Table 9. Results. Two obstacles, 1000 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	190.5	1853.65	38	4.015	4.015
RRT*	1000	1398.4	24	2.4	188.61
RRT* Goal	1000	1366.6	26.5	1.285	158.26
RRT* Limits	1000	1425.4	30	7.22	185.17
RRT* GL	1000	1332.2	27	1.5	187.46

Table 10. Results. Three obstacles, 300 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	264	1324.5	27.5	5.53	5.53
RRT*	300	1435.6	24	4.69	13.36
RRT* Goal	300	1056.6	20	1.25	14.47
RRT* Limits	300	1205.2	25	4.4	17.57
RRT* GL	300	1097.2	23	2.25	26.73

Table 11. Results. Three obstacles, 600 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	264	1324.5	27.5	5.53	5.53
RRT*	600	1331.1	23	4.59	44.42
RRT* Goal	600	1065.5	20	1.21	60.28
RRT* Limits	600	1156.9	25	6.06	65.4
RRT* GL	600	1040.2	21	1.43	91.58

Table 12. Results. Three obstacles, 1000 iterations.

Algorithm	# Iterations	Total Cost	# Segments	Goal Time	Total Time
RRT	264	1324.5	27.5	5.53	5.53
RRT*	1000	1137.2	20	2.95	165.3
RRT* Goal	1000	1029.3	21	1.35	189.73
RRT* Limits	1000	1073.8	25	10.84	220.16
RRT* GL	1000	1001.7	21	1.61	270.71

Figures 11–13 show a bar chart of the three experiments with 600 iterations, respectively, to visually compare the resulting values of the tested algorithms.

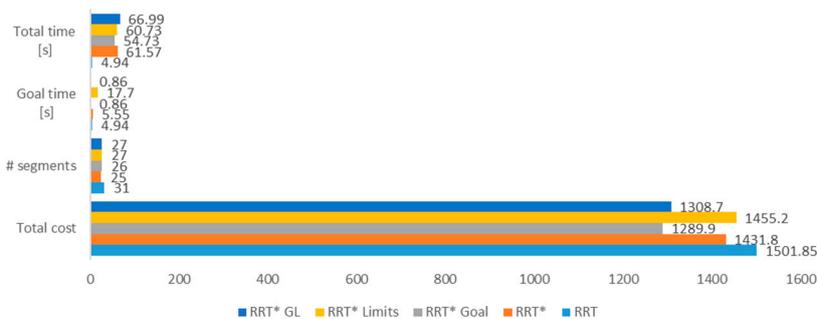


Figure 11. Results of the algorithms for the one obstacle scenario and 600 iterations.

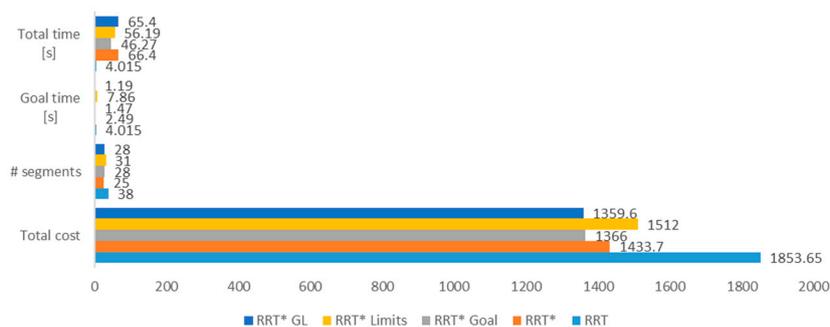


Figure 12. Results of the algorithms for the two obstacles scenario and 600 iterations.

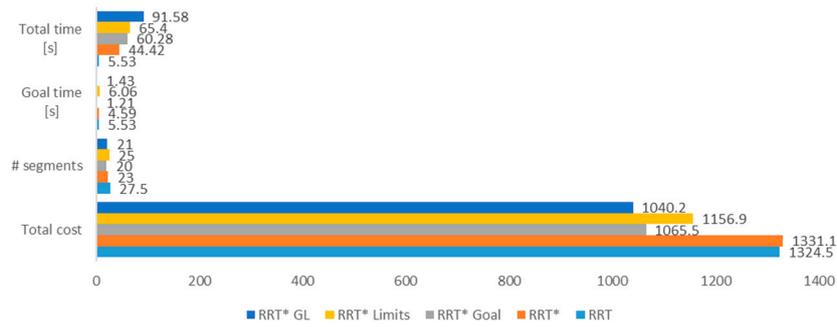


Figure 13. Results of the algorithms for the three obstacles scenario and 600 iterations.

As we can see in the results tables, the RRT* Goal algorithm significantly decreases the time of finding a feasible solution (Goal time), providing more time and computational resources to the path optimization. The RRT* Limits algorithm increases the density of the tree branches around the path, as seen in Figure 14d, which results in lower path costs with respect to the standard RRT*. The combination of both methods shows a significant improvement in the performance of the path planning algorithm, producing a more optimal path without increasing the number of iterations. However, this method requires more time and therefore more computational resources to run. This disadvantage can be balanced by decreasing the number of iterations to obtain similar path costs in less time.

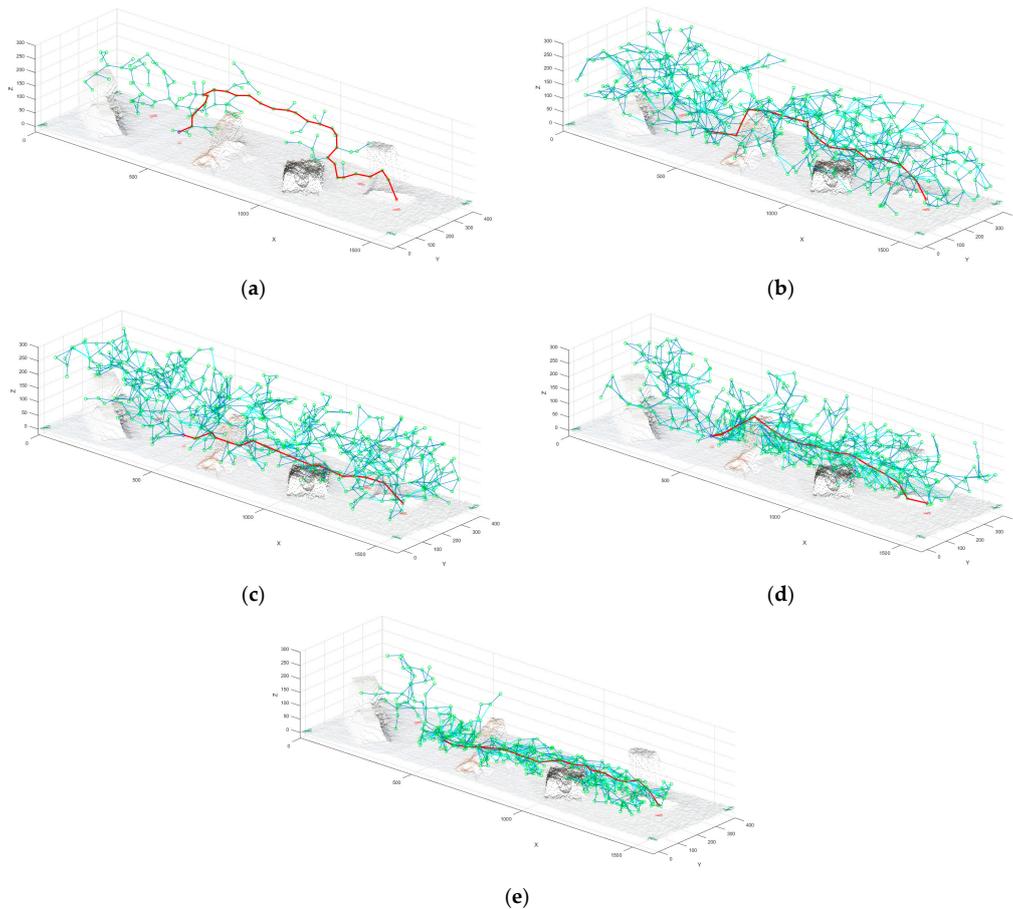


Figure 14. Comparison between RRT* algorithms in a simulation example (600 iterations) in a workspace with three obstacles between the start and goal point. The edges of the tree are presented in blue. The best path that reaches the goal is highlighted in red. (a) The RRT standard; (b) the RRT*; (c) the RRT* Goal; (d) the RRT* Limits and (e) the RRT* GL.

The number of segments of the path in the RRT* is the lowest in most of the cases, which means that in order to achieve a more optimal path, the algorithms need more segments to reduce the path cost.

Video results are provided in [39].

7. Conclusions

In this paper, we experimentally tested the optimal and robust performance of our system including point clouds alignment, target point detection and path planning algorithms.

Flexibility and energy efficiency are important features for the autonomous navigation of robots in a workspace. In our approach, flexibility is given by the automatic alignment of the Kinect V2 point cloud to the robot coordinate frame, using the ICP algorithm. Target point detection also provided flexibility to the system. The color based threshold technique was sufficient for obtaining the goal region centroid of each target point. This approach showed a high performance with the RGB source of the Kinect.

Several variations of the RRT algorithm were successfully tested in order to compare their results with respect to the autonomy and energy efficiency of robots' navigation. We have selected RRT* GL, which achieves a fast and optimal path between the start and goal point. The RRT* Goal algorithm reduces the time required for finding a feasible solution. The RRT* Limits algorithm improves the optimization by increasing the density of the tree branches in the path region.

Acknowledgments: This work is part of the project "Perception and Localization system for autonomous navigation of rotor micro aerial vehicle in GPS-denied environments (VisualNav Drone)" from the Research Center CICTE, directed by Wilbert G. Aguilar. The research assistant Stephanie G. Morales thanks for the encouragement and technical support during the testing of this work to Energypetrol S.A.

Author Contributions: Wilbert G. Aguilar directed the research; Wilbert G. Aguilar and Stephanie G. Morales conceived and designed the experiments; Stephanie G. Morales implemented and performed the experiments; Wilbert G. Aguilar and Stephanie G. Morales analyzed and discussed the results. Both authors wrote and revised the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Koren, Y. *Robotics for Engineers*; McGraw-Hill: New York, NY, USA, 1998.
2. Cabras, P.; Rosell, J.; Pérez, A.; Aguilar, W.G.; Rosell, A. Haptic-based navigation for the virtual bronchoscopy. *IFAC Proc. Vol.* **2011**, *44*, 9638–9643. [[CrossRef](#)]
3. Aguilar, W.G.; Angulo, C. Estabilización de vídeo en micro vehículos aéreos y su aplicación en la detección de caras. In Proceedings of the IX Congreso de Ciencia y Tecnología ESPE, Sangolquí, Ecuador, 28–30 May 2014.
4. Gonzalez, R.; Safabakhsh, R. Computer vision techniques for industrial applications and robot control. *Computer* **1982**, *15*, 17–32. [[CrossRef](#)]
5. Aguilar, W.G.; Angulo, C. Robust video stabilization based on motion intention for low-cost micro aerial vehicles. In Proceedings of the 11th International Multi-Conference on Systems, Signals & Devices (SSD), Barcelona, Spain, 11–14 February 2014.
6. Vasishth, O.; Gigras, Y. Path planning problem. *Int. J. Comput. Appl.* **2014**, *104*, 17–19. [[CrossRef](#)]
7. Henry, P.; Krainin, P.; Herbst, E.; Ren, X.; Fox, D. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In Proceedings of the 12th International Symposium on Experimental Robotics (ISER), Delhi, India, 18–21 December 2010.
8. Thrun, S.; Burgard, W.; Fox, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 24–28 April 2000.
9. Gutmann, J.-S.; Fukuchi, M.; Fujita, M. 3D perception and environment map generation for humanoid robot navigation. *Int. J. Robot. Res.* **2008**, *27*, 1117–1134. [[CrossRef](#)]

10. Oliver, A.; Kang, S.; Wunsche, B.; MacDonald, B. Using the Kinect as a navigation sensor for mobile robotics. In Proceedings of the Conference on Image and Vision Computing New Zealand, Dunedin, New Zealand, 26–28 November 2012.
11. Benavidez, P.; Jamshidi, M. Mobile robot navigation and target tracking system. In Proceedings of the 6th International Conference on System of Systems Engineering, Albuquerque, NM, USA, 27–30 June 2011.
12. Rao, D.; Le, Q.; Phoka, T.; Quigley, M.; Sudsang, A.; Ng, A.Y. Grasping novel objects with depth segmentation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 18–22 October 2010.
13. Ali Shah, S.A.; Bennamoun, M.; Boussaid, F. A novel algorithm for efficient depth segmentation using low resolution (Kinect) images. In Proceedings of the IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), Auckland, New Zealand, 15–17 June 2015.
14. Liu, J.; Yang, J.; Liu, H.; Tian, X.; Gao, M. An improved ant colony algorithm for robot path planning. *Soft Comput.* **2016**, *1*, 1–11. [[CrossRef](#)]
15. Glasius, R.; Komoda, A.; Gielen, S.C.A.M. Neural network dynamics for path planning and obstacle avoidance. *Neural Netw.* **2000**, *8*, 125–133. [[CrossRef](#)]
16. Xin, D.; Hua-hua, C.; Wei-kang, G. Neural network and genetic algorithm based global path planning in a static environment. *J. Zhejiang Univ. Sci. A* **2005**, *6*, 549–554.
17. Seraji, H.; Howard, A. Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Trans. Robot. Autom.* **2002**, *18*, 308–321. [[CrossRef](#)]
18. Kuffner, J.J.; LaValle, S.M. RRT-connect: An efficient approach to single-query path planning. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 24–28 April 2000.
19. Devaurs, D.; Thierry, S.; Cortés, J. Efficient sampling-based approaches to optimal path planning in complex cost spaces. In *Algorithmic Foundations of Robotics XI*; Springer: Cham, Switzerland, 2015; pp. 143–159.
20. Gammell, J.D.; Srinivasa, S.; Barfoot, T. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), Chicago, IL, USA, 14–18 September 2014.
21. Lachat, E.; Héline, M.; Tania, L.; Pierre, G. Assessment and calibration of a RGB-D camera (Kinect V2 Sensor) towards a potential use for close-range 3D modeling. *Remote Sens.* **2015**, *7*, 13070–13097. [[CrossRef](#)]
22. Fankhauser, P.; Bloesch, M.; Rodriguez, D.; Kaestner, R.; Hutter, M.; Siegwart, R. Kinect v2 for mobile robot navigation: Evaluation and modeling. In Proceedings of the 2015 International Conference on Advanced Robotics (ICAR), Istanbul, Turkey, 27–31 July 2015.
23. Hatledal, L.I. Kinect V2 SDK 2.0—Colored Point Clouds. Available online: <http://laht.info/kinect-v2-colored-point-clouds/> (accessed on 20 November 2015).
24. Aguilar, W.G.; Angulo, C. Real-time video stabilization without phantom movements for micro aerial vehicles. *EURASIP J. Image Video Proc.* **2014**, *1*, 1–13. [[CrossRef](#)]
25. Aguilar, W.G.; Angulo, C. Estabilización robusta de vídeo basada en diferencia de nivel de gris. In Proceedings of the VIII Congreso de Ciencia y Tecnología ESPE, Sangolquí, Ecuador, 6–8 June 2013.
26. Myronenko, A.; Song, X. Point set registration: Coherent point drift. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 2262–2275. [[CrossRef](#)] [[PubMed](#)]
27. Eggert, D.W.; Lorusso, A.; Fisher, R.B. Estimating 3-D rigid body transformations: A comparison of four major algorithms. *Mach. Vis. Appl.* **1997**, *9*, 272–290. [[CrossRef](#)]
28. The MathWorks Inc. Pcregrigid Documentation. 2015. Available online: <http://www.mathworks.com/help/vision/ref/pcregrigid.html> (accessed on 24 February 2016).
29. Navon, E.; Miller, O.; Averbuch, A. Color image segmentation based on adaptive local thresholds. *Image Vis. Comput.* **2005**, *23*, 69–85. [[CrossRef](#)]
30. Bruce, J.; Balch, T.; Veloso, M. Fast and inexpensive color image segmentation for interactive robots. In Proceedings of the IEEE/RSJ International Conference on intelligent Robots and Systems, Takamatsu, Japan, 31 October–5 November 2000.
31. Sahoo, P.K.; Soltani, S.; Wong, A.K.C. A survey of thresholding techniques. *Comput. Vis. Graph. Image Proc.* **1988**, *41*, 233–260. [[CrossRef](#)]
32. Heinz, K.; Hanson, W. Interactive 3D segmentation of MRI and CT volumes using morphological operations. *J. Comput. Assist. Tomogr.* **1992**, *16*, 285–294.

33. Corke, P. *Robotics, Vision & Control: Fundamental Algorithms in Matlab*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 73.
34. Aarts, E.H.L.; Lenstra, J.K.; Wiley, J.; Johnson, D.; McGeoch, L.A. The traveling salesman problem: A case study in local optimization. *Local Search Comb. Optim.* **1997**, *1*, 215–310.
35. Karaman, S.; Frazzoli, E. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Robotics Science and Systems VI 104*; The MIT Press: Cambridge, MA, USA, 2010.
36. Aguilar, W.G.; Angulo, C. Real-time model-based video stabilization for microaerial vehicles. *Neural Proc. Lett.* **2016**, *43*, 459–477. [[CrossRef](#)]
37. Aguilar, W.G.; Angulo, C.; Costa, R.; Molina, L. Control autónomo de cuadricópteros para seguimiento de trayectorias. In Proceedings of the IX Congreso de Ciencia y Tecnología ESPE, Sangolquí, Ecuador, 28–30 May 2014.
38. Aguilar, W.G.; Angulo, C. Compensación y aprendizaje de efectos generados en la imagen durante el desplazamiento de un robot. In Proceedings of the X Simposio CEA de Ingeniería de Control, Barcelona, Spain, 1–2 March 2012.
39. Aguilar, W.G.; Morales, S.G. RRT Optimal for UAVs. Available online: <https://www.youtube.com/watch?v=Fe3AACHkcdQ> (accessed on 12 July 2016).



© 2016 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (<http://creativecommons.org/licenses/by/4.0/>).