Abstract—This paper presents a local path planning method for real time applications with Unmanned Aerial Vehicles in unknown environments by considering an onboard sensor of a limited range. Both static and mobile obstacles are considered. The collisions are detected from a laser sensor readings and an efficient algorithm computes the solution trajectory to avoid them. The proposed approach generates a visibility graph and then a Dijkstra's algorithm is run to compute the shortest path. The execution time obtained is low because the proposed approach adds few nodes to model both static and mobile obstacles. The algorithm has been integrated in ROS (Robot Operating System) framework. Many simulations have been performed in different scenarios with a dynamic quad-rotor model to test the proposed algorithm.

Index Terms—Unmanned Aerial Vehicles, local path planning, collision avoidance.

I. INTRODUCTION

Unmanned Aerial Vehicles are becoming more and more important in a lot of applications such as surveillance [1], structure assembly [2], fire detection and monitoring [3], etc. Planning algorithms compute paths and UAV should follow that predefined path during the execution. However, unexpected events can take place when the environment is unknown. For example, static or mobile obstacles can appear. Moreover, the onboard sensors could limit the perception of the environment. In these cases, typical methods to follow the paths usually cannot guarantee to track a predefined path in a safe way. To address these possible situations, a local path planning method should be implemented in order to react when a potential collision is detected.

Other important issue is related to the computational load to resolve the collisions detected in real time. In general, planning approaches spend the same effort on free space as in space with obstacles. Paths should be only computed when it is needed, that is, when the goal is not reachable. Moreover, collision avoidance should be very efficient to ensure the real time requirement.

This paper presents a local path planning method in unknown environments by considering an onboard sensor of a limited range. The method has a low computational load and is suitable for real time applications. It builds a graph when a static or mobile obstacle should be avoided such that it models the obstacle to avoid the collision. The basis is to generate a simple graph only when the goal is not reachable. Thus, the computation time to generate a safe path is very low.

The main contribution of the proposed method with respect to the published works is to consider both mobile and static obstacles in real time applications.

The algorithm has been tested in Gazebo 1.0, which is integrated in ROS [22] Fuerte. Realistic simulations have been performed in different scenarios with a dynamic quad-rotor model based on the implemented in the Hector-quadrotor ROS package [4].

The paper is organized into six sections. Related works are presented in Section II. The description of the problem addressed is shown in Section III. The proposed method
is described in Section IV. Section V presents the simulations performed in Gazebo. Finally the conclusions are detailed in Section VI.

II. RELATED WORKS

Many works have been published in the literature to address the problem of local path planning. Most works on path planning are not good candidates because a low computational load is needed to compute the solution in real time. Among these methods can be highlighted: Rapidly-exploring Random Trees (RRT) [5], probabilistic roadmaps [6], evolutionary techniques [7] [8] [9], particle swarm optimization [10], multi-objective evolutionary algorithms [11], graph search like A* and D* [12], etc.

Planning approaches for UAVs are classified in [13] and [14]. In general, path planning methods can be divided into three groups: cell decomposition methods [23], roadmap methods [15] [16] and artificial potential methods [17] [18] [19]. Cell decomposition methods divide the configuration space into a number of regions. Each cell is allocated a label to identify if it is or not collision free. Thus, a local path planning algorithm finds a sequence of collision free neighbouring cells in order to ensure a feasible path. Roadmap methods build a network of collision free paths from nodes generated in the space. Among these methods highlight visibility graphs and Voronoi diagrams. The methods based on potential fields computes collision free paths considering forces defined as the negative gradient of a potential function to avoid the obstacles. Attractive forces are defined to move toward the goal. These methods presented a disadvantage because the solution can fall into local minima.

The method presented in this paper falls under the category of generating a visibility graph like representation and searching over it. Following this features, an interesting work is presented in [20]. The approach called SPARTAN is successful in real time applications with a quad-rotor operating outdoors considering static obstacles.

III. PROBLEM FORMULATION

The problem considered in this paper is the local path planning of an UAV to perform the missions proposed by the ARCAS project\(^1\). The ARCAS FP7 European Project is developing a cooperative free-flying robot system for assembly and structure construction (see Figure 1). The ARCAS system will use aerial vehicles (helicopters and quad-rotors) with multi-link manipulators for assembly tasks [21]. The aerial vehicles carry structure parts that will be assembled at the target destination. An important part in ARCAS is cooperative assembly planning and safe trajectory generation assuring that neither the aerial vehicles nor the manipulators or the objects carried collide with each other. Therefore, a local path planning algorithm should be implemented to ensure that each vehicle executes a safe path when unexpected events take place.

Our goal is to compute a path for an UAV as the shortest path to the final waypoint while avoiding static obstacles or others UAVs (mobile obstacles). A cost function related to the distance travelled should be minimized and a constraint on the turn angle velocity is considered.

It is assumed that velocity of the vehicle is constant and heading changes are allowed to solve the conflicts. Therefore, the solution

\(^1\)http://www.arcasproject.eu
considers the addition of intermediate waypoints to the trajectory. The information that the method needs is the following:

- Initial flight plan
- Model of the UAV
- Location of the UAV

Other details are taken into account in the problem:

- The environment is unknown
- The sensor range is limited, so the obstacles are detected on the fly
- The planning frequency is 5Hz

The objective is to follow the initial flight plan while obstacles are not detected so that the computational load drops. The efforts are focused on the area where obstacles are detected and should be avoided to follow the initial flight plan.

IV. DESCRIPTION OF THE METHOD IMPLEMENTED

This section describes the proposed approach to address the problem considered. First, a map is built from the laser sensor readings. Then a visibility graph is generated from the detected obstacles. Finally, a safe trajectory is computed in order to reach the goal or final waypoint.

A. Map building

A local map centered on the robot is built from the readings. They are a set of distances and angles which origin is the robot’s position. The space is discretized, so it is divided into two-dimensional cells. Six types of cells are considered (see Figure 2): free, obstacle, virtual obstacle, node, start and goal. Free cells are considered when the readings are the maximum value of distance. Cells containing readings less than the maximum value are marked as obstacle cells. Then, obstacle cells are expanded radially a distance given by the minimum safety distance. The free cells encompassed in this expansion are defined as virtual obstacle cells. These cells are technically equivalent to obstacle cells because they must not be visited. Finally, node cells are added around the virtual obstacle cells, containing nodes of the visibility graph. Thus, the node cells surrounding the obstacles are created. Start and goal cells are the cells where initial and final waypoints are defined, respectively.

![Fig. 2. Example of map building and types of cells considered.](image)

The map resolution, number of cells per meter, can be adjusted and influences the algorithm’s behaviour. Better resolution will usually return better trajectories, but execution time will increase (see Table I). Therefore, a trade-off should be performed to adjust this parameter. A simulation has been carried out to show the different trajectories that arise when surrounding an obstacle with different number of cells per meter (see Figure 3). The blue trajectory is obtained when a cell per meter is considered, the red one when nine cells are considered and the dotted black trajectory when twenty-five cells are considered. Note that the solution trajectory depends on the map resolution.

B. Local path planning with static obstacles

The proposed local path planning algorithm considers several strategies that adapt well to unknown environments and ensure safe trajectories (see Algorithm 1). First, it
TABLE I
EXECUTION TIME DEPENDING ON THE MAP RESOLUTION.

<table>
<thead>
<tr>
<th>Number of cells per meter</th>
<th>Execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0529 ± 0.0447</td>
</tr>
<tr>
<td>9</td>
<td>0.0541 ± 0.0423</td>
</tr>
<tr>
<td>25</td>
<td>0.0583 ± 0.0448</td>
</tr>
<tr>
<td>36</td>
<td>0.0601 ± 0.0460</td>
</tr>
</tbody>
</table>

Fig. 3. Different solution trajectories considering the same flight plan and varying the map resolution (number of cells per meter).

receives the laser sensor readings and checks if a straight path is feasible to fly to the next waypoint of the flight plan from the current location considering the motion constraints. If that waypoint can be reached without detecting collisions a visibility graph is not built. Therefore, execution time is very low when navigating through open spaces.

If straight path is not feasible, two possibilities can be applied. The first one is to keep the current destination waypoint if there is no frontal obstacle at close range. This could be useful for navigation through corridors.

The second one is chosen when a collision needs to be avoided. This is the more important block of the proposed algorithm. It is a graph search based on Dijkstra’s algorithm. First, the visibility graph should be generated with the node cells modeling the obstacles detected and including start and goal cells. The straight line joining two nodes is computed to build an arc. If the straight line passes through either obstacle or virtual obstacle cells, no connection is considered between both nodes. If the connection is feasible, the weight of the arc is set to the euclidean distance between the nodes. The only case when the weight considered is different happens when the arc links the start node with another node. Then the weight considered is a linear combination of distance and deviation from the current course. Once the graph is built, the Dijkstra’s algorithm is run. This represents the greater computational load of the algorithm.

Algorithm 1 Navigation algorithm

if goal reached then
    Update goal
else
    Map building
    if Straight path is feasible then
        Waypoint ← Goal
    else
        if No obstacles ahead then
            Keep course
        else
            Graph search
            if Path found then
                Waypoint ← Next node in the path
            else
                Go around
            end if
        end if
    end if
end if

The number of nodes can be reduced depending on the kind of obstacles. This is done to decrease the execution time. At first, node cells are marked in the map so that they can be used in the algorithm. However, when finding long straight obstacles, only the extreme cells are considered as nodes. This considerably decreases the execution time but it is also less accurate.

The algorithm might not be able to compute a feasible path. In that case, our last choice is to go around the obstacle. Although this case rarely happens, the problem
is solved by looking for nearby nodes and minimizing course deviation in order to go around the obstacle.

The proposed algorithm (see Algorithm 1) ensures to navigate through very dense unknown environments and can be applied in real time.

C. Local path planning with mobile obstacles

This problem presents important difficulties such as: to detect the mobile obstacle, estimate the trajectory of the mobile obstacle and compute the maneuver to avoid the collision (see Algorithm 2).

Obviously, the detection is performed from the sensor reading but a matching should be done to identify the kind of obstacles, static or mobile. Each reading is compared with its neighbour readings. If an abrupt change is measured, the system saves a new obstacle. Every obstacle is stored and defined by a list of the readings that form the obstacle. Once all obstacles have been detected, they should be identified as static or mobile. This matching is essential to detect mobile obstacles. The decision on the obstacle’s mobility lies on the estimated displacement of the center of the obstacle.

**Algorithm 2** Mobile obstacle avoidance algorithm

Obstacles detection
Centers computation
Match each obstacle with the ones seen previously
if Obstacle is moving then
   Extrapolation
   if Collision detected then
      Predict possible own trajectories
      Choose the safest one
   end if
   Erase obstacle from readings
end if

Mobile obstacles can disappear for some instants. In order to avoid successive new identifications, the readings describing an obstacle are stored for a half second after disappearing. This is also useful as both the sensor and the mobile obstacle are not always on the same horizontal plane. If a disappeared obstacle appears again, the previous data could be used to identify it.

Once a mobile obstacle is detected, its trajectory should be estimated to detect potential collisions. A collision takes place when the separation between two vehicles is less than one meter. The trajectory is extrapolated linearly by minimum squares considering three previous measures of the obstacle position. This is the only information used in the extrapolation. It is important to point out that mobile obstacles are erased from the map and are only taken into account when a collision is detected. Thus, the addition of nodes is avoided as an obstacle is not going to provoke a collision.

Finally, mobile obstacle avoidance should be efficiently computed for real time applications. The goal is to add an intermediate waypoint between the current position and the next waypoint so that the UAV can avoid the collision. The computation of this waypoint is based on a set of waypoints that are layed out at a certain distance of the current position of the robot. This set meets the kinematics constraints of the vehicle. A trajectory is simulated for each waypoint. Figure 4 shows two possible intermediate waypoints and their trajectories after detecting a potential collision between two vehicles.

The choice of the best intermediate waypoint depends on the minimum distance between mobile obstacle and the vehicle considered in any time. In the next iteration, the vehicle checks if it can reach the next waypoint. If it is not possible, it checks if it can safely follow the intermediate waypoint computed in the previous iteration. If that is not feasible either, another intermediate waypoint should be computed to avoid the collision detected. This new intermediate waypoint substitutes the one computed in the
previous iteration.

Fig. 4. Collision prediction and evaluation of safest waypoint considering a mobile obstacle (in red).

V. SIMULATIONS IN ROS

Many simulations have been carried out in a realistic environment that simulates the multi-UAV testbed of the CATEC (Centro Avanzado de Tecnologías Aeroespaciales) which is equipped with a VICON localization system that provides estimation of the position of the vehicles with a precision of few millimeters in real time (see Figure 5). The simulation of aerial vehicles in complex indoor environments is done by using Gazebo and RViz has been used as a visualizer in order to validate the algorithm visually. Moreover, the algorithm has been integrated in ROS framework with the same ROS node architecture used in the multi-UAV testbed of CATEC. This makes the transition between simulation and experimentation straightforward as it diminishes the possible faults in the transition. The dimensions of the scenario are $15 \times 15 \times 4$ m.

The proposed algorithm has been run in a HP ProBook 4520s PC equipped with an IntelTM Celeron P4500 processor (1.87GHz), 4 GB of RAM and Intel HD 4000 graphical card. The operating system used was Ubuntu 12.04 Linux. The code was written in C++ language and integrated with ROS Fuerte distribution. The dynamic quad-rotor model used is based on the Hector-quadrotor ROS package [4]. The Hokuyo UTM-30LX laser sensor is placed upside-down under the vehicle and is used to detect the obstacles. The laser sensor provides measures in 2D dimensions. For this reason 2D problems are considered in this section.

Scenarios with static and mobile obstacles have been considered to test the proposed algorithm. Simulations have been computed in the same machine. The videos show the performance of the algorithm in real time. The videos of the different performed simulations are available at http://www.youtube.com/0grvc0.

The map resolution is $4 \text{cells/m}^2$, the minimum safety distance considered is $1$ m (i.e. 2 cells), the velocity is $0.5$ m/s, and the minimum and maximum turn angle velocity are $-0.7 \text{rad/s}$ and $0.7 \text{rad/s}$, respectively. Since the planning frequency is $5$ Hz, the

Fig. 5. Multi-UAV testbed placed in the CATEC facilities.

Fig. 6. RViz visualization of testbed simulated by Gazebo.
trajectory will be computed every 0.2s.

A. Dense static map

The algorithm has been run in several dense unknown environments with static obstacles. A set of simulations is presented in this section to show the behaviour of the vehicle in this kind of scenarios. The parameters which have been changed in this set of simulations are the velocity, which values are 0.5, 1.0 or 1.5m/s, and the obstacles of the environment. Three cases are considered (see Figure 7): the first one considers all the obstacles; the second one considered all the obstacles excepted the labeled one as 1; and, the third one removes the obstacles labeled as 1 and 2. The combination of parameters which have been tested in this simulations, along with the execution times obtained, are shown in Table II.

Figure 7 shows the trajectories flown at 0.5m/s. The flight plan is defined by a sequence of waypoints (numbered green circles) and the quad-rotor has to fly the plan while avoiding the obstacles of the environment. The solution trajectory for the first case is shown in blue line. The two corridors surrounding obstacle 1 are too narrow to be visited without risk because of the security distance imposed. Consequently, the algorithm decides to surround the triangular obstacle.

The histogram of execution times per iteration of the algorithm for the simulation when all the obstacles are considered (first case) and $v = 0.5m/s$ is shown in Figure 8. These times are always under the 0.2s which is the limit imposed by the 5Hz planning frequency. This proves that the algorithm is very suitable for real time applications and can be run in each iteration to compute a flight plan in order to reach the goal.

The solution trajectory computed in the second case is shown in red line. The algorithm computes a different trajectory and it moves the vehicle between obstacle 2 and the triangular obstacle. Finally, the green line shows the solution trajectory without obstacles 1 and 2. Obviously, this is the shortest path of the three displayed, because there is more free space available and therefore less risk of collision. The execution times to plan a path are shown in Table II and different videos are available in our mentioned Youtube channel.

![Fig. 7. Solution trajectories computed in dense unknown environments with static obstacles (in blue).](image)

![Fig. 8. Histogram of the execution time per iteration for full map simulation. Velocity is 0.5m/s.](image)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$v = 0.5m/s$</th>
<th>$v = 1.0m/s$</th>
<th>$v = 1.5m/s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>43±39ms</td>
<td>43±39ms</td>
<td>43±38 ms</td>
</tr>
<tr>
<td>Without 1</td>
<td>44±38ms</td>
<td>42±37 ms</td>
<td>45±41 ms</td>
</tr>
<tr>
<td>Without 1 and 2</td>
<td>40±37 ms</td>
<td>40±37 ms</td>
<td>47±39 ms</td>
</tr>
</tbody>
</table>

The histogram of execution times per iteration of the algorithm for the simulation when all the obstacles are considered (first case) and $v = 0.5m/s$ is shown in Figure 8. These times are always under the 0.2s which is the limit imposed by the 5Hz planning frequency. This proves that the algorithm is very suitable for real time applications and can be run in each iteration to compute a flight plan in order to reach the goal.
B. Mobile obstacle avoidance

Several simulations with two quad-rotors have been performed. One of them should change its trajectory when detecting a collision, and the another one does not change its trajectory. Simulations are run considering different trajectories and the duration is on ten minutes to test the reliability and safety of the algorithm in this kind of scenarios.

Figure 9 depicts a simulation performed. Each quad-rotor fly its trajectory during on seven minutes. Seven collisions are detected and all are solved. Quad-rotor QR1 adds intermediate waypoints to its flight plan to avoid the collision with quad-rotor QR2. The safety of the simulation is presented in Figure 10 where the separation between both quad-rotors is shown in blue. The same simulation has been carried out without considering obstacle avoidance and the result is shown in green line. When running the simulation with no obstacle avoidance, both quad-rotors violate the safety distance at $t = 24s$ and they finally collide at $t = 40s$.

How a collision is avoided by QR1 (white points) is shown in Figure 11. Four instants are presented to see the maneuver performed by QR1. The potential collision is represented by a red circle and the intermediate waypoint computed by QR1 is represented by a green circle.

The execution times of the algorithm considering mobile obstacles are shown in Figure 12. The mean time is 0.041s and the standard deviation is 0.037s. Therefore, the algorithm is very suitable for real time applications and can be run in each iteration. Moreover, this time is less than the one obtained considering static obstacles as fewer nodes are generated in the visibility graph when it is needed.
VI. CONCLUSION

A local path planning algorithm considering Unmanned Aerial Vehicles in unknown environments has been presented. The algorithm computes a solution trajectory for a vehicle considering static obstacles and the rest of non-cooperative vehicles as mobile obstacles. The main advantage of the proposed algorithm is its low computational load.

The map building to model both static and mobile obstacles plays an important role in the proposed approach because it minimizes the number of nodes in order to generate the visibility graph. This influences the execution time and ensure that the algorithm can be applied in real time. Then, a Dijkstra’s algorithm is implemented to compute the shortest collision-free path.

The results obtained improve the ones presented in [20]. The work presented in this paper considers mobile obstacles and the execution times and standard deviations obtained in both scenarios with static or mobile obstacles are less than the ones obtained with the SPARTAN method, 0.071s and 0.087s respectively.

Future work will consider environments with more vehicles and other kind of maneuvers to avoid collisions such as change of velocity and/or altitude will be implemented. Furthermore, real experiments will be carried out in order to validate the algorithm.

ACKNOWLEDGMENT

This work has been supported by the ARCAS Project, funded by the European Commission under the FP7 ICT Programme (ICT-2011-287617) and the CLEAR Project (DPI201 1-28937-C02-01), funded by the Ministerio de Ciencia e Innovacion of the Spanish Government. The authors would like to thank David Alejo for his help with technical implementation.

REFERENCES


