Evaluation of Reactive Obstacle Avoidance Algorithms for a Quadcopter

Cyrille Berger, Piotr Rudol and Mariusz Wzorek
Linköping Univeristy
SE-581 83 LINKÖPING, SWEDEN
firstname.lastname@liu.se

Alexander Kleiner
iRobot, Pasadena, CA
akleiner@irobot.com

***Abstract*—In this work we are investigating reactive avoidance techniques which can be used on board of a small quadcopter and which do not require absolute localisation. We propose a local map representation which can be updated with proprioceptive sensors. The local map is centred around the robot and uses spherical coordinates to represent a point cloud. The local map is updated using a depth sensor, the Inertial Measurement Unit and a registration algorithm. We propose an extension of the Dynamic Window Approach to compute a velocity vector based on the current local map. We propose to use an OctoMap structure to compute a 2-pass A* which provide a path which is converted to a velocity vector. Both approaches are reactive as they only make use of local information. The algorithms were evaluated in a simulator which offers a realistic environment, both in terms of control and sensors. The results obtained were also validated by running the algorithms on a real platform.**

I. Introduction

For successful deployment of Unmanned Aerial Vehicles (UAV) into a civil airspace, it is necessary to guarantee that they can operate safely in the environment. This requires robust collision avoidance algorithms and that UAVs can navigate using only proprioceptive sensors and algorithm should run on board.

There are two main categories of algorithms that solve the problem of obstacle avoidance and navigation of an autonomous vehicle, path planning and reactive algorithms. Path planning techniques compute a globally optimal path in the environment, such as the A* algorithm [8]. When combined with a globally consistent map, path planning algorithms can compute an optimal solution. Path planning algorithms are computationally intensive for ground robots [12], which have fewer degrees of freedom than an UAV. For UAVs, the main challenge is the explosion of the search space dimensionality, to address this problem a random set of nodes can be generated and then checked for collision [16], the position of the nodes can be updated dynamically [10].

Reactive techniques only consider local information and attempt to steer the vehicle away from obstacles while aiming towards the goal. Reactive techniques have been applied for many types of vehicles, such as ground robots [6], boats [1] or airplanes [14], where the motion model of the vehicle has to be taken into consideration. Using only current sensor data, the Dynamic Window Approach (DWA) [5] use the motion model of the vehicle to select the best trajectories among the controllable and safe ones.

In practice, it is desirable to combine a path planning approach with a reactive one. The robot can move along a path that guarantees it will reach its goal. The reactive technique would be able to guarantee to avoid local obstacles.

Both types of obstacle avoidance techniques require the detection of obstacles, which can be realized with a variety of sensors. A comparison between vision-based and Lidar approaches can be found in [11]. For UAVs, vision is commonly used, such as in [15], where a Kalman Filter is used to track and estimate the size of obstacles. It is possible to use a 2D Lidar [3] and to apply the same kind of path planning techniques for UAVs than for ground robot. The drawback is that this does not allow to detect obstacles that are outside of the scanning plane of the sensor.

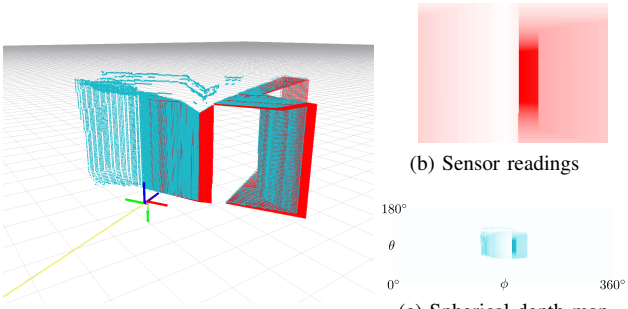
Since reactive obstacle avoidance techniques do not require accurate localisation, they are the more suited to guarantee safe navigation for quadcopters. For this work, small quadcopters, around 0.7m in diameter, equipped with a 3D depth sensor and with limited computational power, are considered.

Based on those considerations, the obstacle avoidance algorithm cannot rely on a globally consistent map, but it can use a local map. The main contributions of this paper include an efficient algorithm for local map construction and representation and two obstacle avoidance algorithms. The local world model is based on the idea of a spherical depth and the fusion of IMU information with depth sensor using a modified Iterative Closest Point (ICP) technique [4]. The first obstacle avoidance algorithm is inspired by DWA [5], which we adapted to quadcopters. The second one is a local 2-pass A*, using an OctoMap [9] representation which allows to get a structure for computing the algorithm efficiently.

Section 2 of this paper presents the construction of the local world model and how it is updated with new sensor readings. In section 3 and 4 the two obstacle avoidance algorithms are presented, respectively. In section 5 we present simulated experimental results as well as the validation on an actual platform. The significance of those results are discussed in the last section.

Both obstacle avoidance algorithm assume that the robot is

This work is partially supported by the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT network organization for Information and Communication Technology, and the Swedish Foundation for Strategic Research (CUAS Project, SymbiKCloud Project).



(a) 3D view (b) Sensor readings (c) Spherical depth map

Fig. 1: This figure show the 3D view (1a), the sensor readings (1b) and the corresponding spherical depth map (1c). In blue the point clouds from the local map, in red the point cloud from the sensor readings.

currently at position O and trying to reach a goal G. d_{max} is the maximum depth provided by the depth sensor.

II. Local Spherical Depth Map

To compensate the narro field-of-view (FOV) of the depth sensor (less than 60° horizontal), past information is fused with current information in a local *spherical depth map*. The map is centred around the robot. The point cloud is stored using spherical coordinates, in a 2D matrix (see fig. 1), where the values of the matrix $\mathcal{M}$ are the depth while rows and columns correspond to the polar and azimuth angle:

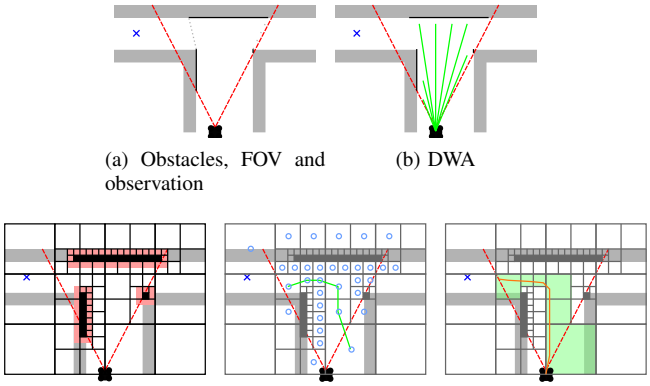
$$\mathcal{M}(idx(\theta, \pi, s_p), idx(\phi, 2\pi, s_a) = depth(\theta, \phi) \tag{1}$$

s_p and s_a specify the resolution of the depth image on the polar axis and the azimuth axis and idx is a function that computes an integer index from a floating point angle:

$$idx(x, m, s) = \frac{x \bmod m}{m} s \tag{2}$$

This representation is very quick to query for obstacle and for fusion but does not handle occlusion. For each sensor reading, the local map is updated with the following algorithm:

1) The first step is to compute the transformation $T_{t-1,t} = (R_{t-1,t}, \vec{t}_{t-1,t})$ (where $R_{t-1,t}$ is the rotation and $\vec{t}_{t-1,t}$ is the translation) between the map at the previous time step $\mathcal{M}_{t-1}$ and the point cloud $\mathcal{S}_t$ coming from the current sensor data at instant t.
   The rotation $R_{t-1,t}$ is initialised with the IMU data from the quadcopter. Then a simple variant of the ICP algorithm, using a Levenberg-Marquardt optimisation, is used to estimate the translation $\vec{t}_{t-1,t}$ and to improve the estimation of the rotation.
2) The transformation $T_{t-1,t}$ is used to provide an initial estimate of the map $\mathcal{M}_t$ for the current time step:
$$\mathcal{M}_t = T_{t-1,t} \cdot \mathcal{M}_{t-1} \tag{3}$$
3) The points from S_t are projected in the spherical coordinate and replace the value from the local map $\mathcal{M}_t$



(a) Obstacles, FOV and observation (b) DWA (c) Octomap divisions and non-flyable area (d) First A* on the end-nodes of the OctoMap (e) Second A* with higher accuracy but limited to the end-nodes selected in the first A*

Fig. 2: Diagram explaining the concept behind the dynamic window and 2-pass A* approach. For simplification reasons, the diagrams show the concepts in 2D but the algorithms are working in 3D. In all figures, the blue cross represent the goal, the grey lines are the obstacles, the black lines are the perceived obstacles, the black dot represents the quadcopter and the red lines show the FOV. In fig. 2b, the green line shows the admissible velocities. In fig. 2c, the grid shows the octomap structure, the black squares are the obstacles, the red squares are the no-fly zone. In fig. 2d, the blue circle indicates admissible nodes in the OctoMap and the green polyline shows the result of the A* algorithm. In fig. 2e, the green rectangles represent the area where the second A* is computed. The red polyline is the final path.

III. Dynamic Window Approach

DWA was introduced in [5] for ground robots whose admissible trajectories are circular arcs. The robot is controlled with its curvilinear velocity and its angular velocity. Quadcopters have more degrees of freedom and are controlled with a velocity vector. Given the current velocity vector $\vec{v}_t = (\theta_0, \phi_0, \dot{\rho})$, in spherical coordinates. The admissible controllable velocity vectors are given by:

$$\vec{v}_i = v_i \cdot \vec{d}_i \tag{4}$$

$$|\vec{v}_t - \vec{v}_i| < \delta_t \cdot a_{max} \tag{5}$$

Where δ_t is the time interval between two sensor readings frame, a_{max} is the maximum acceleration of the quadcopter and $\vec{d}_i$ is the direction vector of the velocity. For each admissible direction $\vec{d}_i$, DWA computes the best velocity v_i^b and an heuristic $value(\vec{d}_i)$ of the quality of that direction. The next controlled velocity is then selected with:

$$\exists j, \vec{v}_{t+1} = v_j^b \vec{d}_j \mid \forall i, value(\vec{d}_i) \leq value(\vec{d}_j) \tag{6}$$

a) Computation of distance to obstacle: For a given direction $\vec{d}$, given a point on an obstacle pt and its projection $p_{\vec{d}}(pt)$ on the line $(O, \vec{d})$, $dto(pt, \vec{d})$ is the distance to the

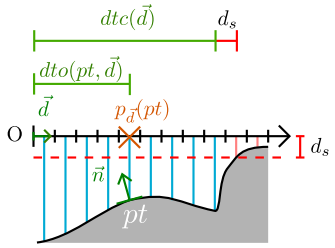


Fig. 3: Measurements used to evaluate a direction $\vec{d}$ in DWA. The gray area represent an obstacle. $\vec{n}$ is the normal of the obstacle surface, $dtc(\vec{d})$ is the distance to collision from the quadcopter to the first obstacle in the safety distance (d_s).

obstacle pt, along the direction d:

$$dto(pt, \vec{d}) = dist(p_{\vec{d}}(pt), O) \tag{7}$$

Where $dist(A, B)$ is the euclidean distance between the point A and B. The quadcopter will collide with an obstacle pt, if $dto(pt, \vec{d}) < d_s$, where d_s is the safety distance. The distance to collision $dtc(\vec{d})$ along the direction $\vec{d}$ is given by:

$$dtc(\vec{d}) = \min_{pt|dto(pt,\vec{d})<d_s} (dist(p_{\vec{d}}(pt), O)) - d_s \tag{8}$$

b) Computation of the best velocity: The maximum admissible velocity is the maximum that the quadcopter can reach and still be able to stop before colliding with the obstacle. It is given by the following equation:

$$v_{max}^2 = dtc(\vec{d}) a_{max} + \frac{v_t^2}{2} \tag{9}$$

Flying at the maximum admissible velocity can be dangerous since it would require to be highly reactive to stop before reaching the obstacle. Instead, a lower speed is selected, which depends on the distance to collision:

$$v_{best} = v_{max} \frac{dtc(\vec{d})}{d_{max}} \tag{10}$$

c) Evaluation of safety of a direction : To check if a direction $\vec{d}$ is going closer to an obstacle or if it is going further away, the angle of the direction $\vec{d}$ with the normal vector of each obstacles can be used (see fig. 3). That value is then averaged:

$$\Sigma_n = \frac{1}{n} \sum \vec{n} \cdot \frac{\vec{OG}}{\|\vec{OG}\|} \tag{11}$$

Lets name Σ_o the average distance between all the obstacles and a direction. Then the maximum of that value is $\max(\Sigma_o)$.

d) Computation of the value of a direction: The following measurements are computed:

- h (goal directness), a measurement of how much the vector $\vec{d}$ is in the direction of the goal:

$$h = \frac{\vec{OG}}{\|\vec{OG}\|} \cdot \vec{d} \in [0, 1] \tag{12}$$

- r, a measurement of how closer to the goal the quadcopter can be if following that direction:

$$r = \max(1, \frac{dtc(\vec{d})}{dist(O, G)}) \in [0, 1] \tag{13}$$

- n, a measurement of whether the quadcopter is moving closer to the obstacle or further away:

$$n = 1 - \Sigma_n \in [0, 1] \tag{14}$$

- a_o, a measurement of how far this trajectory is from the obstacles:

$$a_o = \frac{\Sigma_o}{\max(\Sigma_o)} \in [0, 1] \tag{15}$$

- tti, a measurement of the time to impact, if the quadcopter move at velocity v_{best} in the direction $\vec{d}$:

$$tti = \frac{dtc(\vec{d})}{d_{max}} \frac{v_{min}}{v_{best}} \in [0, 1] \tag{16}$$

- kh, a measurement of how much the direction $\vec{d}$ is aligned with the current heading of the quadcopter, given ψ the current heading and $\psi(\vec{d})$ the heading of $\vec{d}$:

$$kh = cos(\psi - \psi(\vec{d})) \in [0, 1] \tag{17}$$

- s (speed maximization), a measurement of how fast the quadcopter can go in the direction $\vec{d}$:

$$s = \frac{v_{best}}{v_{max}} \in [0, 1] \tag{18}$$

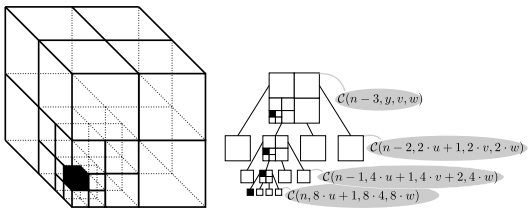
The value of a direction is given by:

$$\begin{aligned} value(\vec{d_i}) =& \alpha h^2 r^2 + \beta n^2 + \gamma a_o^2 + \delta s^2 \\ &+ \epsilon tti^2 + \zeta kh^2 \end{aligned} \tag{19}$$

The result of that function is in the range $[0, 1]$, a value closer to 1 will make the direction $\vec{d_i}$ more likely to be selected by the algorithm. The parameters α, β, γ, δ, ϵ and ζ are constants weights that select the importance given to each measurements and have been selected empirically. We selected for our test platform the following values $\alpha = 0.3$, $\beta = 0.3$, $\gamma = 0.15$, $\delta = 0.05$, $\epsilon = 0.05$, $\zeta = 0.15$. α and β are the highest, since α encourages the quadcopter to move towards the goal while β ensure that the quadcopter moves away from obstacles. γ controls how close to the obstacle the quadcopter flies and ζ discourages change of direction, it is set at an average value, since a high value would prevent the quadcopter to turn while a low value would encourage the quadcopter to zigzag. δ controls how fast the quadcopter can go while ϵ controls how quickly the quadcopter would reach an obstacle.

IV. Two-passes A*

Hierarchical A* has been used to solve global path planning for large scale problems [7]. It sacrifices the guarantee of optimality in exchange for a significant reduction in computational cost. Considering the size of the problem, a 2-pass A* was used, the first pass is done using a very coarse model, with a very limited number of nodes, and the result is used to restrict the search area in the second A* pass.



(a) 3D View of an OctoMap (b) Corresponding tree

Fig. 4: The black cube correspond to a cell that has been marked as an obstacle.

a) Octomap representation: The main benefits of the OctoMap [9] structure is that it is very fast to query for obstacles and to compute distance to obstacle and it provides a subdivision of the space into large *open* area, see fig. 2c, which reduces the search space dimension for the A* algorithm. An OctoMap covering a cube of length l_0 was used, the robot is located at coordinate $(0, \frac{l_0}{2}, \frac{l_0}{2})$, so that the OctoMap covers the cube in front of the robot and it covers the FOV of the depth sensor.

b) Notations: $\mathcal{C}(k,u,v,w)$ is a cube in the OctoMap at level k and with the index (u,v,w) it covers the cubic space from coordinates $(u\cdot l_k, v\cdot l_k, w\cdot l_k)$ to $((u+1)\cdot l_k, (v+1)\cdot l_k, (w+1)\cdot l_k)$ where $l_k = \frac{l_0}{2^{(k-1)}}$.

In the hierarchy, if $k \neq 0$, $\mathcal{C}(k,u,v,w)$ has parent $\mathcal{C}(k-1, \lfloor\frac{u}{2}\rfloor, \lfloor\frac{v}{2}\rfloor, \lfloor\frac{w}{2}\rfloor)$ and if $k \neq n$, it can have the following eight children: $\mathcal{C}(k+1, 2\cdot u+a, 2v+b, 2\cdot w+c)$ with $(a,b,c) \in \{0,1\}^3$. The notations can be seen in fig. 4.

c) OctoMap construction: The OctoMap is constructed by iterating over all points in the local map, and marking the cube in the lowest level as an obstacle. A cube $\mathcal{C}(k,u,v,w)$ is instantiated, if only if, $\exists pt \in \mathcal{M}_t$ such as $pt \in \mathcal{C}(k,u,v,w)$.

d) A from node to node:* Lets call end-nodes, the nodes of the OctoMap that do not have children. To get a rough estimate of the path, the A* algorithm is used on the end-nodes of the OctoMap which are at least partially in the FOV of the sensor, as shown in fig. 2d.

Two safety distances are used, d_{ss} and d_s. d_{ss} defines an area where the UAV is absolutely forbidden to go, while d_s defines a distance that the UAV should avoid, if possible. A node is considered to be *admissible* if it is in the FOV of the sensor and if at least one point of that node is at a distance superior to d_{ss} from any obstacle. Two nodes are *compatible* if they have a common corner in the FOV. If they are compatible, they can be considered as part of the same path. The following cost function between two nodes (the source node, n_1, and the destination node, n_2) is used:

$$cost(n_1,n_2) = \max(1, \frac{d_{max}-b}{dto(n_2)-b}) \cdot dist(n_1,n_2) \quad (20)$$

$$b = 2\cdot d_s - d_{max}1 \quad (21)$$

$dto(n_2)$ is the distance to the closest obstacle of n_2. The path found by the A* on the end-nodes of the OctoMap is denoted as:

$$\mathcal{P}_1 = \{\mathcal{C}(k_i,u_i,v_i,w_i)/i \in [0,p]\} \quad (22)$$

e) Second A pass on the set of nodes:* the path given by the A* on the end-node of the OctoMap is not usable for navigation. The second A* pass is restricted to the volume of the environment given by the path $\mathcal{P}_1$:

$$\mathcal{V}(\mathcal{P}_1) = \bigcup_{i\in[0,p]} \mathcal{C}(k_i,u_i,v_i,w_i) \quad (23)$$

That volume is subdivided in cube of size $l_n = 0.15m$, which constitutes the node for the A* search. Like for the first A* pass, a node is *admissible* if it is in the FOV of the sensor and if the centre of that node is at a distance superior to d_{ss} of any obstacle. The same cost function (20) is used. The second A* pass generates a path $\mathcal{P}_2 = \{p_0,\ldots,p_q\}$. This path $\mathcal{P}_2$ is used to compute a velocity vector that can be used to control the quadcopter. Given the r first points of $\mathcal{P}_2$ which form a line, the velocity is selected with the following equation:

$$\vec{vel} = \frac{\overrightarrow{p_0p_q}}{\|\overrightarrow{p_0p_q}\|} \cdot \min(v_{max}, \frac{\|\overrightarrow{p_0p_q}\|}{4}) \quad (24)$$

Where v_{max} is a maximum velocity. When no path is found with the first or second pass, the UAV turn around on itself until it finds a path.

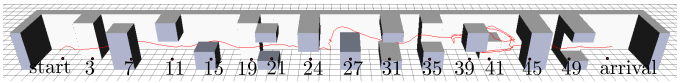
Fig. 5: LinkQuad platform and the Asus Xtion Pro sensor used for experimental validation.

V. Experimental Validation

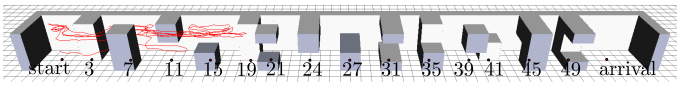
We have conducted experimentation with a simulation and with a real platform. During our experimentation no collision between an UAV and an obstacle occurred. The results shown in this section are to evaluate how well a UAV could perform using exclusively our collision avoidance algorithms for navigation to perform challenging missions.

We are targeting an internally developed autonomous UAV (see fig. 5) of 0.70m radius. It can carry a range of different sensors, such as the Asus Xtion Pro depth camera with a maximum range of 10m, with an horizontal FOV of $57°$ and a vertical one of $45°$.

For the kind of sensor installed on our platform, $d_{max} = 10m$. Therefore we used $l_0 = 10m$ for the OctoMap size. The number of levels n of the OctoMap defines the accuracy of the model. In our experiments we have used a value of $n = 7$, which correspond to a size of $l_n = 0.15m$ for the lowest levels in the OctoMap.



(a) A*



(b) DWA

Fig. 6: Trajectories with 60° FOV and ICP for a $4m$ wide corridor.

FOV	Experiment		Average time	Best distance reached	Count
60°	A*	ICP	632s	56m	3
		AL	656s	56m	1
		SV	877s	49m	1
	DWA	ICP	373s	7m	4
		AL	627s	29m	3
		SV	872s	15m	2
120°	A*	ICP	403s	56m	4
		AL	384s	56m	4
		SV	500s	56m	1
	DWA	ICP	690s	25m	1
		AL	728s	15m	1
		SV	891s	19m	2

TABLE I: Corridor experiment. The experiment was run four times. The count is the number of time the UAV reached the maximum distance.

A. Simulation

A simulator based on Gazebo [13] with ROS integration was used to evaluate the algorithms. The same control system and dynamic model as our quadcopter is used. Two types of experiments where conducted to evaluate the presented methods. In the first experiment the UAV has to fly in a corridor filled with obstacles. In the second one it has to reach a number of beacons in any order.

The experiments were conducted using different sensor setups and local map algorithms. We used a simulated sensor with a FOV of 60°, similar to what currently available real depth sensors provide, and of 120°, a FOV that we consider would be more adapted for obstacle avoidance. For the local map algorithm, we use three variants, one using ICP, one with a single view (SV) (the local map of previous time is ignored) and one with accurate localisation (AL). The single view mode is used to evaluate the benefit of using a local map. The accurate localisation mode allows to evaluate the quality loss due to the lack of a good localisation.

1) Corridor: For this experiment, the UAV have to move through a 50 meters long corridor, filled with obstacles, as shown in fig. 6. The experiment stops after 900s.

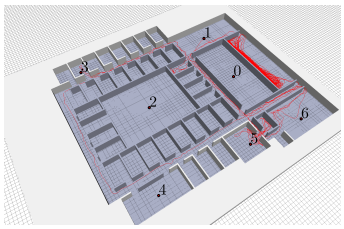
The timing for the various experimental settings are shown in table I.

2) Reaching Beacons: In this experiment a number of beacons are put in the environment, and the UAVs knows the direction and relative distance, for instance it could be WiFi access point detected with signal strength [2].

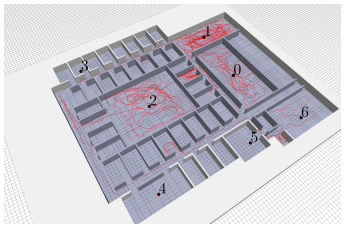
There are seven beacons as shown in fig. 7. The UAV gets 3 minutes to reach a beacon, otherwise it switches to a different beacon, the closest one. The number of times a beacon has been reached is shown in table II.

FOV	Experiment		0	1	2	3	4	5	6
60°	A*	ICP	4	4	4	2	3	3	4
		AL	3	4	2	0	4	4	4
		SV	4	4	1	0	1	4	3
	DWA	ICP	4	4	0	0	0	1	3
		AL	2	1	1	0	1	3	4
		SV	4	4	0	0	1	4	4
120°	A*	ICP	4	4	1	0	3	4	4
		AL	4	4	0	0	1	4	4
		SV	2	3	0	2	0	3	3
	DWA	ICP	3	4	0	0	0	1	2
		AL	1	2	0	0	0	4	1
		SV	4	4	1	0	1	4	0

TABLE II: Number of times a beacon has been reached by the different combinations of algorithm and modes. The experiments were performed four times.



(a) A*



(b) DWA

Fig. 7: Trajectories with 60° FOV and ICP for the beacon reaching experiment.

3) Computational time: On a desktop computer, the update of local spherical map takes around 150ms. DWA needs 10ms to evaluate one direction. In our experimentation, 40 possible directions are evaluated, which means a running time of 400ms. The 2-pass A* has a non constant run time, in average is is under 300ms but it could some times reach up to 6s. A deadline of 300ms was set to get a result from the 2-pass A*. When the deadline is reached the current best path is returned.

B. Experimentation with a real-platform

Experimentation with a real-platform were conducted to validate the simulation.

1) Experiment:

a) Accurate localisation and static obstacle: In the first experiment, accurate localisation is used in combination with a static obstacle. The UAV was successful in avoiding the obstacle and reaching the goal, as seen in fig. 8.

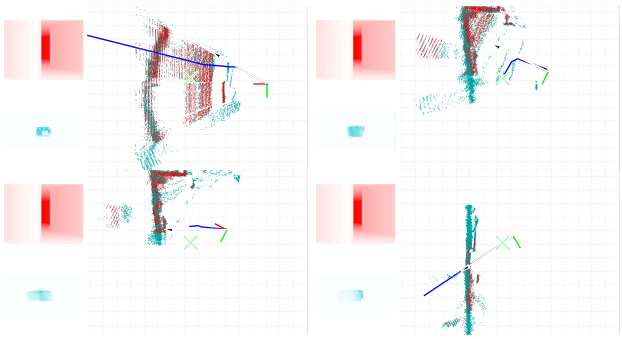


Fig. 8: With accurate localisation and a static obstacle. In each figure, the top left window is the sensor view, the bottom left is the local map. The blue point cloud is the local map and the red one is the sensor. The blue trajectory represent the plan by the 2-pass A*, and the black arrow is the velocity computed from the plan. The green cross represent the goal.

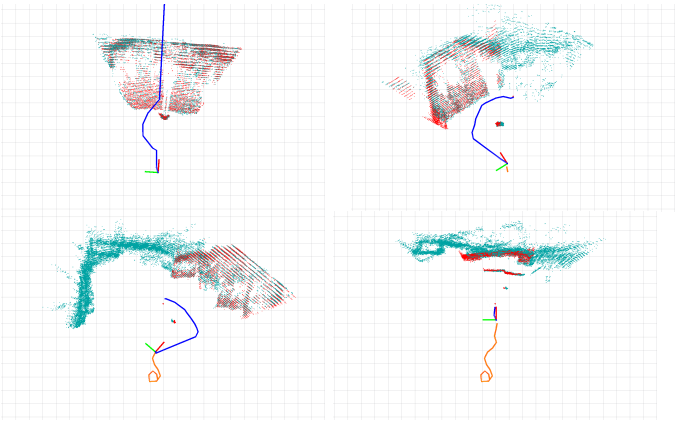


Fig. 9: With inaccurate localisation and a dynamic obstacle. The blue point cloud is the local map and the red one is the sensor. The blue trajectory represent the plane by the 2-pass A*, and the black arrow is the velocity computed from the plan. The orange line represents the path.

b) No accurate localisation and dynamic obstacle: In the second experiment, only IMU localisation is used in combination with a dynamic obstacle. The quadcopter was successful in avoiding the obstacle and reaching the goal, as seen in fig. 9.

VI. Discussion and Conclusion

In this work, we have presented two obstacle avoidance algorithms that use a spherical depth map to control the velocity of an UAV while avoiding obstacles in the environment. The methods that we have presented only use proprioceptive sensors and can therefore be used to guarantee that the UAVs will not collide when flying in the environment.

In all our experiments, as expected, the 2-pass A* behave better than the DWA. The experiments were designed to be challenging, with small distances between obstacles and under such circumstances both approaches demonstrated they would allow a UAV to perform part of its misson.

The result show that, without a local map, a sensor with a FOV of 60° is inadequate for navigation. However, with a FOV of 120°, our algorithms perform nearly as well, with or without the local map.

The experimentations with a real platform have validated that the simulation algorithm can be transposed to a real platform. The quadcopter was successful in avoiding a static obstacle as well as a dynamic obstacle.

Further work will involve the integration of a path planner with the DWA that would then be executed on board of the quadcopter. As DWA can be used to validate and adjust in real time the current velocity generated from the path of the motion planner.

References

[1] T. Bandyophadyay, L. Sarcione, and F. Hover. A simple reactive obstacle avoidance algorithm and its application in singapore harbor. In *Field and Service Robotics*, volume 62 of *Springer Tracts in Advanced Robotics*, pages 455–465. 2010.

[2] J. Biswas and M. Veloso. Wifi localization and navigation for autonomous indoor mobile robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4379–4384, May 2010.

[3] A. Ferrick, J. Fish, E. Venator, and G.S. Lee. Uav obstacle avoidance using image processing techniques. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 73–78, April 2012.

[4] A. W. Fitzgibbon. Robust registration of 2D and 3D point sets. In *British Machine Vision Conference*, pages 662–670, 2001.

[5] D. Fox, Burgard W, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23–33, Mar 1997.

[6] R. Glasius, A. Komoda, and S. C.A.M. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125 – 133, 1995.

[7] D. Harabor and A. Botea. Hierarchical path planning for multi-size agents in heterogeneous environments. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 258–265, Dec 2008.

[8] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[9] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[10] S. Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 807–814. IEEE, 2008.

[11] S. Hrabar. An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance. *Journal of Field Robotics*, 29(2):215–239, 2012.

[12] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *The International Journal of Robotics Research*, 21(10-11):917–942, 2002.

[13] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 7628 of *Lecture Notes in Computer Science*, pages 400–411. 2012.

[14] R. Sharma, J. B. Saunders, and R. W. Beard. Reactive path planning for micro air vehicles using bearing-only measurements. *Journal of Intelligent and Robotic Systems*, 65(1-4):409–416, January 2012.

[15] Y. Watanabe, A. J. Calise, and E. N. Johnson. Vision-based obstacle avoidance for uavs. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, August 2007.

[16] M. Wzorek, G. Conte, P. Rudol, T. Merz, S. Duranti, and P. Doherty. From motion planning to control - a navigation framework for an autonomous unmanned aerial vehicle. In *Proceedings of the 21st Bristol UAV Systems Conference (UAVS)*, 2006.