Aerial Service Vehicles for Industrial Inspection: Task Decomposition and Plan Execution

Jonathan Cacace, Alberto Finzi, Vincenzo Lippiello, Giuseppe Loianno, and Dario Sanzone *

Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli Federico II, via Claudio 21, 80125, Naples, Italy

Abstract. We propose an autonomous control system for Aerial Service Vehicles capable of performing inspection tasks in buildings and industrial plants. In this paper, we present the applicative domain, the high-level control architecture along with some empirical results. The system is assessed on real-world and simulated scenarios representing industrial environments.

Keywords: Aerial Service Robotics, High level Control Architecture, Mixed Initiative Planning and Execution.

We present a high level architecture designed for an Aerial Service Vehicle (ASV) operating in close interaction with the external environment. This work is framed within the The AIRobots project [1] whose aim is to develop a new generation of unmanned service helicopters, equipped with sensors and end-effectors, and capable not only to fly, but also to achieve robotic tasks in proximity and in contact with the surface (e.g. site inspections, simple manipulations, etc.).

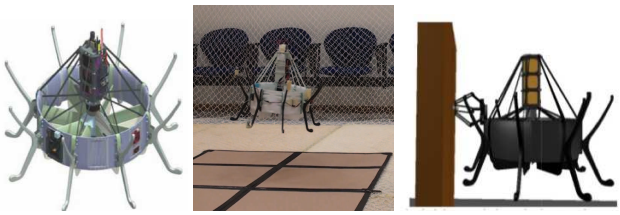

Fig. 1. Robotic Platform: ducted-fan ASV

In our scenario, the autonomous system should orchestrate a new set of operations like wall approach, docking, undocking, wall scanning etc.. These operations represent different operative modes, associated with a different controller with specific control laws and performance the high-level control system should be aware about. Each switch

* The research leading to these results has been supported by the AIRobots collaborative project, which has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement ICT-248669.

from one operative mode to the other should be suitably prepared and planned to keep smooth control trajectories. Since the system flies close to the obstacles in cluttered and unknown environments, fast planning engine are required to generate (or to adjust) trajectories in real-time. On the other hand, the system should be able to regulate the trade off between fast planning and accurateness of the generated trajectories depending on the operative mode and context. Moreover, since the system operates with the man in the loop, the planning and executive system should be able to manage sliding autonomy from autonomous to teleoperated mode depending on the humans' interventions. This applicative domain is challenging and novel and has not been investigated in depth in the UAV literature; this is mainly focused on free flight tasks and simultaneous localization, mapping, and path planning problems [4, 8, 17]. Few high-level architectures for UAV can be found in literature [5], but none of these addresses the complexity of the operative domain proposed in this paper.

1 System Requirements and Architecture

The applicative scenario described so far requires a high-level control system with following features:

- The air vehicle operates in close interaction with the environment, hence reactive, adaptive, and flexible planning/replanning capabilities are needed;
- Both autonomous and human-in-the-loop control should be supported to allow human interventions and teleoperation;
- High-level control strategies should be defined taking into account the low-level operative modes and constraints.

In particular, the high-level system should orchestrate the activations of a set of low-level controllers, modeled as hybrid automata [14], switching to the appropriate low-level controller according to the operative mode and to the desired task (see Fig. 2) while feeding the selected controller with suitable data (e.g. actual state and references signals).

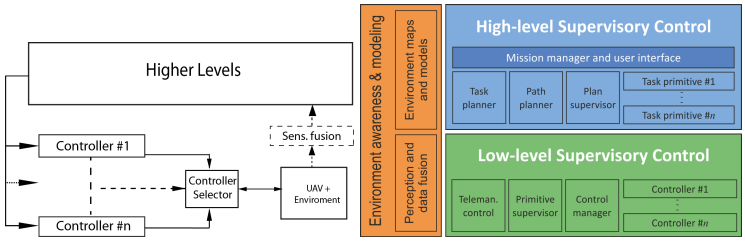

Fig. 2. Interaction between the high level system and the low-level controllers (left); the high level control system is composed of high-level and low-level supervisory systems.

To match these requirements we proposed the layered architecture depicted in Fig. 2 and Fig. 3. Here, two layers are distinguished, the high-level supervisory system is re-

sponsible for user interaction, task planning, path planning, execution monitoring, while the low-level supervisory system manages the low-level execution of control primitives setting the controllers and providing control references. The architecture is detailed in Fig. 3.

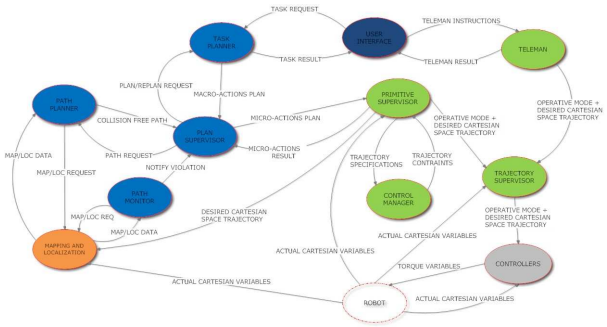

Fig. 3. High Level Architecture: high level, low level, and reactive level modules are respectively in blue, green, and gray

The *User* module (US) allows us to specify high-level goals (e.g. $Inspect(p)$) or lower level tasks (e.g. $TakeOff$) or directly to teleoperate. Each task/goal is delivered to the TP which expands a task into an abstract plan composed of *macro-actions*. This plan is then provided to the *Plan Supervisor* (PS) for high-level execution. Each task or macro-action can be interrupted and pre-empted by new tasks provided by the user, causing to the TP to manage task replanning. The PS generates, for each macro-action in the high-level plan, a set of *micro-actions* to be executed by the *Primitive Supervisor* (PR). Each *macro-action* is further decomposed into a sequence of *micro-actions* which are endowed with detailed information about the geometrical paths. The PR exploits the *Control Manager* (CM) to select the low-level controller responsible for the execution depending on the micro-action. Finally, the PR generates the control trajectory passing it to the *Trajectory Supervisor* (TS) to obtain control references at a suitable frequency. It exploits concatenations of fifth-order polynomials providing smooth trajectories between waypoints [11] while ensuring the velocity and tolerance constraints. When a micro-action fails, the PS can either call the PP to generate an alternative path either call the TP to generate a different plan of macro-actions, furthermore it can be interrupted by the *Path Monitor* (PM) which checks for trajectory deviations and unexpected obstacles. Finally, the operator can always switch to a manual control, in this case TS should monitor the trajectory provided by *Teleman*. Once the autonomous control is restored, a replanning process is needed to recover the previous task.

1.1 Task Planning and Executive Control

The high-level executive system coordinates task decomposition and plan monitoring. The executive system relies on a PRS engine [9] that manages the BDI-like execution cycle [15] and hierarchical task decomposition. The high-level executive system

responds to events generated by US, PS, or TP itself by committing to handle one pending goal, selecting a method from the library, managing the hierachical decomposition to extract/update the macro-actions plan. Once a plan is generated, PS should manage the actual execution of each *macro-action* providing the action result to the TP module. During this executive process, user interventions are treated in a uniform way: at any time the user can interrupt/suspend the current task, or by call the execution of alternative tasks. In this case, the executive system reacts replanning from the current state by selecting alternative methods, hence generating an alternative plan. This enables mixed initiative task planning [2].

1.2 Path Planning and Replanning

The *Path Planner* expands each *macro-action* into a set of *micro-actions* representing a path that respects geometric and operative constraints. The path generation algorithm extends the Rapidly-exploring Random Tree (RRT) algorithm [10] which is particularly suitable in highly unstructured and dynamic domains. In this work, the RRT algorithm generates collision-free paths composed of sequences of waypoints (x, y, z, θ), where (x, y, z) is a point and θ is the yaw. More specifically, it generates paths as sequences of (x, y, z) points in a 3D search space (3D grid map), while the yaw θ is obtained as the direction pointing towards the next waypoint. The generated path should satisfy a set of additional control, safety, and temporal constraints: *Maximum angle* for pitch and yaw; *Minimal distance* from the obstacles (this parameter is also associated with the operative mode and the accuracy of the selected controller); *Maximum Time* for the path generation processes, if the algorithm cannot find a feasible path before the timeout, it should provide the best partial path. Moreover, that RRT path planner can generates several solutions to refine the path, until one of the following conditions are satisfied: *timeout*, i.e. the available time for path planning expires; *interrupt*, i.e. a replanning request or an exogenous event interrupts plan generation; *cost threshold*, i.e. as soon as the current path cost is below a suitable threshold, the generated plan is considered as satisfactory. The path planning refinement process is illustrated in the Algorithm 1 where the path generation process is iterated until the current generated path is not satisfactory. If the $timeout$ occurs before the generation of the first solution, the $solveRTT$ function generates the $path$ that arrives closer to the target.

Algorithm 1 Refine_RRT(q_{init},q_{goal},$threshold$,$timeout$)

```
initialize(path,time);
while ((time < timeout) ∧ (preempted = false) ∧ (pathCost ≥ threshold)) do
   newPath ← solveRRT(q_init,q_goal,timeout);
   if C(newPath) < path then
      path ← newPath;
      pathCost ← C(newPath);
   end if
end while
return path
```

The path cost is defined as follows:

$$\begin{aligned}\mathbf{c}(path) =& \mathbf{c}_{lng}(path) \cdot p_{lng} + \mathbf{c}_{ang}(path) \cdot p_{ang} + \\ & \mathbf{c}_{way}(path) \cdot p_{way} + \mathbf{c}_{obs}(path) \cdot p_{obs} + \mathbf{c}_{unk}(path) \cdot p_{unk}\end{aligned} \tag{1}$$

where the p_i are suitable weights and $\mathbf{c}_i$ are defined as follows. $\mathbf{c}_{lng}(path)$ is a cost associated with the path length; $\mathbf{c}_{ang}(path)$ represents the cost associated with angular (yaw and pitch) variations, by minimizing this cost a straight path should be preferred to a path with angular variations; $\mathbf{c}_{way}(path)$ keeps track of the generated waypoints and allows us to minimized the segments in the path; $\mathbf{c}_{obs}(path)$ is associated with obstacle proximity and penalizes paths close to obstacles; $\mathbf{c}_{unk}(path)$ penalizes paths through -or close to- unexplored cells. Once a path is generated, the path planner defines a set of constraints $cst = (ms, md, et)$ to each generated segment. Roughtly, for each segment, we set the maximum speed ms directly proportional to the obstacle minimal distance mo along the corresponding segment; ms is also associated with a proportional error et, therefore we set md as mo-et (if this value is not positive, the speed limit is lowered). These constraints cst are also accessible by the human operator which can manually reset them. Note, that cst are just rough limits used by CM and PR to select the right controller and to generate the trajectory associated with the path.

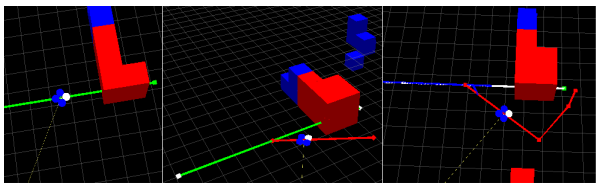

Fig. 4. (left) *Brake* to avoid collision; (center) *Escape* path to avoid the obstacle; (right) *Replan* a new path generated to reach the target.

Path replanning is managed with different strategies depending on the time available for path generation. The urgency associated with the replanning activity depends on the position of the collision point p_{obs} and the estimated time to collision t_{ttc}. This one is estimated considering the obstacle distance d_{obs} along the trajectory and the mean velocity v_{mean} along the path. Given the time to collision t_{ttc}, we introduce two thresholds $T_b < T_e$ used to distinguish the following three cases:

- *Brake.* If $t_{ttc} \leq T_b$ then the obstacle is too close for replanning, hence the PS directly sends a $Brake$ command to the PR to stop the robot in $hovering$ (Fig. 4 up-left).
- *Escape.* If $T_b < t_{ttc} \leq T_e$, the PP is invoked by the PS to find an escape path that allows the robot to avoid the obstacle; the escape trajectory represents fictituos detour that provides the planner with additional time to generate the new path on-the-fly (Fig. 4 up-right).
- *Replan.* If $t_{ttc} > T_e$ then the time is sufficient for safe replanning, hence the PS calls the PP to replan, on-the-fly, a trajectory from a suitable deviation point along the previous path (Fig. 4 down).

The PP is called in the case of *Escape* and *Replan*. In the case of *Escape*, the path planning task is simple, it is to select a close and safe target point q_{target} in the free space, far enought to allow safe on-the-fly replanning, and generate a path to reach it (Fig. 4 right). That is, *Escape* provides a path that not only permits to avoid the obstacle, but also provides the time for replanning a new path to the goal. The interesting case is the third one, where the path planning process should find an alternative path that connects the old trajectory with a new one while the robot is flying. The replanning algorithm is illustrated in Algorithm 2. Given the target q_{goal}, the old path $path_{old}$, the collision point q_{obs}, and the t_{ttc} time, the replanning process first estimates the time needed to replan t_{rp} (*estimatedRepTime*); then it selects a waypoint wp_{rp}, along the old path $path_{old}$, from which it is possible to safely calculate the deviation $path_{new}$ from $path_{old}$ (*selectDeviationWP*); finally, upon setting a suitable threshold (*setThreshold*), the replanning process calls RRT_refine to generate the new path $path_{new}$ from the deviation waypoint wp_{rp} to the target q_{goal}. $path_{new}$ should allow PR to generate a new trajectory connecting the old one with a smooth deviation from wp_{rp}.

Algorithm 2 Replan(q_{goal}, $path_{old}$, q_{obs}, t_{ttc})

$q_c \leftarrow$ getPosition();
$t_{rp} \leftarrow$ estimatedRepTime(q_c, q_{goal}, $path_{old}$, q_{obs});
$wp_{rp} \leftarrow$ selectDeviationWP(q_c, q_{obs}, $path_{old}$, t_{rp});
$threshold \leftarrow$ setThreshold(wp_{rp}, q_{goal}, t_{rp}, t_{ttc});
$path_{new} \leftarrow$ Refine_RRT(wp_{rp} , q_{goal}, $threshold$, t_{rp});
return $path_{new}$

To select the deviation waypoint wp_{rp} we defined the following strategy. Given the estimated time needed to replan t_{rp}, we estimate the robot position q_{pr} at time t_{rp} (assuming that it keeps following the old path $path_{old}$ during replanning), if there exists a waypoint wp in $path_{old}$ that follows p_{rp} and precedes q_{obs} (keeping a suitable range the we assume as $maxRange$), then we select wp as the deviation waypoint wp_{rp}, otherwise, q_{rp} is on the path segment that intersects the obstacle, hence we select wp_{rp} as the point q_m in the middle of the segment that connects q_{rp} and a point q'_{obs} which is at $maxRange$ distance from q_{obst}. In Fig. 4 (center), we find an example of replanning from a waypoint after the collision detection (left).

1.3 3D Mapping

The environment for mapping and path planning is a 3D grid-map of cells run-time generated given the robot pose and 3D point clouds extracted from the cameras. We deploy the well known pin-hole camera model [7]. Pose estimation of the UAV is needed to identify the 3D position of the projected camera points in the world reference frame. Our pose is either obtained by using libviso2 [6] coupled with a Kalman filter or, alternalively, by directly deploying an optitrack motion capture system. Given the pose, the associated point cloud map should be suitable processed into a 3D occupancy grid. This is obtained by discretizing the vehicle's workspace with elementary cubes of equal

size. In our case, we employed a vehicle of $50 \times 50 \times 20$ cm hence, we used cubes of 10 cm. For each cube we stored: number of inliers (3D triangulated points) fell into the cube volume, last camera position which an inlier had been collected and finally the state of the cube. The number of inliers represents the number of different points from which the same obstacle has been detected. The last camera position is required in case of hovering to avoid that the same image features could generate dome wrong inliers, while it is possible that the same outlier is achieved from different points of view. Finally, the state of the cube is used by the path planner to decide to replan the map and choose a safety distance of navigation from the occupied cell. Each cell can be associated with one of the following values: $free$, $occupied$, $obstacle$, $target$, $ignored$ or $unknown$. Initially each cube is set to $free$. When a 3D point is detected to belong to a given cube, the value of the corresponding cube is set to $occupied$. When the number of points internal to a cube reaches a given treshold, the state is set to $obstacle$. On the other hand, when a target is identified the corresponding cube is set to $target$. Moreover, since the camera can observe directly the target from each position that had generated a valid target view point, all the cubes laying along these optical rays are set to $ignored$. For wide environments, a sparse representation of the occupancy grid map is considered together with a spatial/temporal vanishing criterion. This last determines whether an occupancy cube is reliable yet or if it has to be discarded based on the distance travelled by the vehicle or on the time interval spent after its last update. In fact, due to the drift of the vehicle pose estimation, obstacles which have been observed far from the current position or from long time cannot be considered reliable in the current map representation, and they have to be refreshed again. With these solutions the reliability and scalability of the map representation can be tuned with respect to any environment.

2 Experimental Results

In this section, we present experimental results on planning, replanning, and obstacle avoidance, both in real-world scenarios and in simulated environments.

Real-world planning and execution. Our architecture has been tested in a real scenario of dimension $400 \times 400 \times 300\ cm^3$ considering the two environments depicted in Fig. 5 (up and down). In the two tests, the task was the following: inspect a target point in pose $(380, 350, 50, 90)$ from the pose $(40, 40, 50, 0)$ with maximal and minimal speed set at $0.3\, m/s$ and $0.1\, m/s$ respectively. The obstacles are detected on the fly and this can provoke task/path replanning, escape, or brake. For each scenario, we tested 10 times collecting mean, max, min, and standard deviation (STD) of: time spent during planning (Tp), time spent in replanning (Tr), number of replanning episodes (Nr), length of the executed path (Lp), and total time for execution (including replanning time) (Te). For computation and simulation we used an Intel Core Duo, 1.40GHz, 3GB ram, Ubuntu 10.04. The high-level architecture was developed in ROS. For 3D mapping, we used cameras ueye with hardware synchronized images, compressed on-board using atom 1.6 GHz, and sent to a ground station. The stereo images are streamed at around 15 Hz at the ground station. The vision algorithm can track around 120 image features correspondences on 4 images working at the streaming frequency. Each camera provides images with resolution of 752×480 and an angle of view of around $50°$.

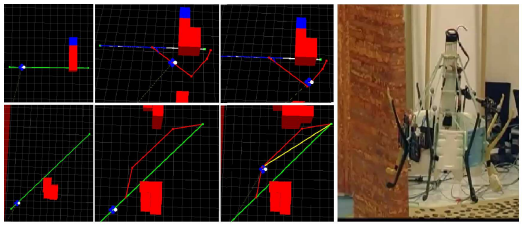

Fig. 5. Replanning: generated and executed path (left) real platform during plan execution (right).

	Test 1				**Test 2**			
	Mean	**STD**	**Max**	**Min**	**Mean**	**STD**	**Max**	**Min**
Tp	0.075	0.014	0.08	0.04	0.017	0.002	0.03	0.01
Tr	0.614	0.41	1.20	0.01	0.067	0.04	0.11	0.005
Te	60.5	10.12	75	42	49.9	8.18	60	40
Lp	14.4	1.54	18	12	13.18	1.11	15	11

Table 1. Planning and execution results (in seconds) in the real scenario.

Tab. 2 reports the results for the two scenarios (Test 1 and Test 2 in Fig. 5). For both these settings, initially the obstacles are not visible, hence we have a low time for planning associated with a simple generated plan (Fig. 5 (left)). Then, replanning is needed to adjust on-line the trajectory once the obstacles are discovered. Replanning and execution time are slightly higher in the first scenario which is more complex. Instead, Tr seems negligible if compared with Te. The final trajectory length (Te) is similar in both the settings and comparable with the distance between the starting and target point, hence the final trajectory seems not affected by continuous replanning. In these tests, Tp and Tr are mainly due to path and trajectory planning (task planning is negligible). We never experienced brake or escape. Overall, the system task/path planning performance seems compatible with the operative scenario requirements.

Simulated planning and execution. We tested our planning and execution system in simulated environments. To test continuous replanning, we considered a larger space of dimension $100 \times 100 \times 50\ m^3$ with 4 and 9 obstacles. To test replanning decoupled from map bulding, we assumed a know map associated with a visibility horizon (not visible obstacles are detected on the fly causing replanning). For each test, the task was to inspect a target point in pose $(90, 90, 5, 90)$ starting from $hovering$ in the pose $(5, 5, 5, 0)$ (in meters); the robot maximal and minimal velocity was set at $0.5\,m/s$ and $0.1\,m/s$ respectively. By changing the visibility horizon (green cells in Fig. 5) of the planner (15 or 25 m) and the complexity of the environment (4 or 9 obstacles) we obtained 4 scenarios.

Tab. 2 collects means and STD of 10 tests for each entry (time and length are in $sec.$ and m, LL, HL, etc. are for Low complexity and Low visibility, High complexity and Low visibility). We can see that Tp increases with the obstacles (HL,HH) and decreases with short visibility (LL,HL). Indeed, in these cases the planning problem is simpler, however, short visibility is associated with additional replanning time which, in turn,

Res/Env	LL		LH		HL		HH	
	Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
Tp	0.21	0.11	0.39	0.03	0.25	0.10	0.31	0.14
Tr	0.12	0.03	0.07	0.01	0.20	0.04	0.23	0.03
Te	308.39	3.1	211.88	2.4	718.57	5.2	720.45	7.6
Lp	79.09	13.76	78.04	9.63	86.79	12.65	85.24	13.12
Nr	0.9	0.21	0.3	0.12	3.4	1.71	2.5	1.10

Table 2. Planning and execution results(time in seconds, length in meters)

decreases with the number of obstacles. The lower the replanning time, the lower is the execution time and the shorter the executed path. A similar effect is due to visibility: short visibility causes frequent replanning events (Nr) and longer paths (Lp) and execution times (Te). Furthermore, the variance is enhanced with short visibility that enhances the uncertainty. In these tests task planning time is usually negligible (Tp and Tr mainly due to path and trajectory). Also in this case, we never experienced brakes or escapes.

Simulated inspection. As for operations closer to the surface, we considered two typical inspection scenarios: physical (Pi) and visual inspection (Vi). In both these cases the system has to move in a pose which faces a vertical surface hovering at a close and fixed distance (approach), in this case 50 cm. As for Pi (see Fig. 6, left), the robot executes a docking maneuver (docking) and slides (keeping the contact) along a linear trajectory (p-inspect) of $225\ cm$. In the case of Vi an inspection trajectory (v-inspect) should be planned and executed. Here, the goal is to scan a $150 \times 100\ cm^2$ surface with step $50\ cm$ distant 50 cm from the wall (see Fig. 6, right). In Tab. 2, we collect the results of 10 tests for each scenario considering planning time (Tp) divided in trajectory (Tm) and path planning (Tpp) time (task planning is negligible).

	Physical Inspection				Visual Inspection			
	Mean	**STD**	**Max**	**Min**	**Mean**	**STD**	**Max**	**Min**
Tp	0.798	0.012	0.019	0.009	0.734	0.47	1.25	0.42
Tm	0.324	0.17	1.07	0.12	0.329	0.22	0.57	0.3
Tpp	0.473	0.27	0.71	0.14	0.405	0.07	0.49	0.39

Table 3. Physical inspection and visual inspection

For each test and scenario, both path and task planning times are compatible with the operative scenario requirements.

3 Conclusions

Aerial Service Robotics is a challenging and novel application for autonomous systems. The close and physical interaction with the environment and the frequent user interventions requires a high-level control system which integrates fast and reactive planning

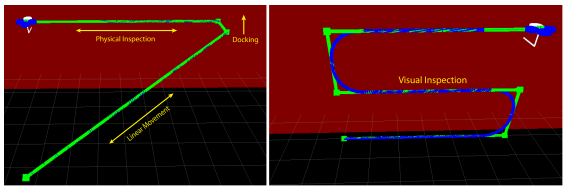

Fig. 6. Physical inspection (left) and visual inspection (right)

engines working at different levels of abstraction and sensitive to low-level operative mode constraints. In this paper, we proposed a system that combines a set of methods showing its performance and feasibility in a industrial plant inspection case study.

References

1. EU Collaborative Project ICT-248669, "AIRobots", *www.airobots.eu*.
2. J. Allen and G. Ferguson, "Human-Machine Collaborative Planning", *NASA Workshop on Planning and Scheduling for Space*, 2002.
3. H. Bay, A. Ess, T. Tuytelaars and L. Van Gool, "Speeded-up robust features (SURF)", *Computer Vision and Image Understanding*, vol.110 no. 3, pp. 346–359, 2008.
4. M. Bloesch, S. Weiss, D. Scaramuzza, and R. Siegwart, " Vision Based MAV Navigation in Unknown and Unstructured Environments", *ICRA*, pp. 21–28, 2010.
5. P. Doherty, J. Kvarnström and H. Fredrik,"A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems", *AAMAS*, pp. 332–377, 2009.
6. A. Geiger, J. Ziegler and C. Stiller, "StereoScan: Dense 3d Reconstruction in Real-time", *IEEE Intelligent Vehicles Symposium*, pp. 963–968, 2011.
7. R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, 2nd ed. Cambridge U.K.: Cambridge Univ. Press, 2004.
8. S. Hrabar," Vision-Based 3D Navigation for an Autonomous Helicopter", *Ph.D. Dissertation, University of S. California*, 2006.
9. F. Ingrand, M. P. Georgeff and A. S. Rao, "An architecture for Real-Time Reasoning and System Control", *IEEE Expert: Intelligent Systems and Their Applications*, pp. 34-44, 1992.
10. S. M. Lavalle,"Rapidly-Exploring Random Trees: A New Tool for Path Planning", *Computer Science Dept., Iowa State University, Tech. Rep*, 1998.
11. S. E. Macfarlane and E. A. Croft,"Jerk-bounded manipulator trajectory planning: design for real-time applications", *IEEE Transactions on Robotics*, vol. 19, pp.42–52, 2003.
12. M. Morales, L. Tapia, R. Pearce, S. Rodriguez and N. Amato, "A machine learning approach for feature sensitive motion planning", *Int. Workshop on the Algorithmic Foundations of Robotics*, 2004.
13. S. Holmes, G. Klein and D.W. Murray, "A square root unscented Kalman filter for visual monoSLAM", *ICRA*, pp. 3710–3716, 2008.
14. R. Naldi, M. Marconi, and L. Gentili, "Modelling and control of a flying robot interacting with the environment", Journal of IFAC, 47(12):2571–2583, 2011
15. A. S. Rao and M. P. Georgeff,"Deliberation and its Role in the Formation of Intentions", *UAI*, pp. 300–307, 1991.
16. Robotics Operating System, "ROS", *www.ros.org*.
17. A. Stentz," Optimal and efficient path planning for unknown and dynamic environments", *Int. J. of Robotics and Automation*, 10(3):89–100, 1995.