POLITECNICO DI TORINO

Dipartimento di Ingegneria Meccanica e Aerospaziale

Corso di Dottorato in Meccatronica

SDSS: ING-INF/01-Elettronica

PhD Thesis

Advances in Human Robot Interaction for Cloud Robotics applications

Advisor

Basilio Bona

Candidato

Gabriele Ermacora

Maggio 2016

XXVIII ciclo

Contents

List of Figures

Abstract

In this thesis are analyzed different and innovative techniques for Human Robot Interaction. The focus of this thesis is on the interaction with flying robots. The first part is a preliminary description of the state of the art interactions techniques. Then the first project is Fly4SmartCity, where it is analyzed the interaction between humans (the citizen and the operator) and drones mediated by a cloud robotics platform. Then there is an application of the *sliding autonomy* paradigm and the analysis of different degrees of autonomy supported by a cloud robotics platform. The last part is dedicated to the most innovative technique for human-drone interaction in the User's Flying Organizer project (UFO project). This project wants to develop a flying robot able to project information into the environment exploiting concepts of Spatial Augmented Reality.

“per aspera ad astra”

Greetings

YES, I am totally aware that the quote "*per aspera ad astra*" is not particularly original for a thesis. It is actually the only perfect quote I could ever find for this thesis. This goes to the people that 10 years ago, when I enrolled in my first class of Mechanical Engineering, were saying " You will never make it" – and now I am totally proud to answer " Not only I made it XXX, but now I'm even a doctor!!!". Therefore I want to thank those people that somehow made me discover my attitude and develop one of my best skills: **the art of being madly stubborn!**

Moreover I really don't know how I would have survived for 10 years in Politecnico di Torino without my family - special thanks to my family not only for supporting me but especially for giving me always a wise suggestion and positive mood in facing challenges. Many thanks to my supervisor Basilio Bona and especially for having sent that e-mail that I received when I was broken and backpacking in Australia and made me become after 4 long years a doctor in robotics. Thank you to the people and academic researchers working in the Robotic Research Group (now JOL CRAB) at the Politecnico di Torino that taught me a lot. Special thanks to Stefano Rosa that supported me and mentored me also in the scientific side and taught me tons of super nerd stuff! Thank you to the TIM researchers working in the JOL CRAB, especially the director Marco Gaspardone that guided me throughout those years and taught me how to make innovation and scientific research coexist. Thank you to Dieter Schmalstieg and Friedrich Fraundorfer and all the UFO team that mentored, stimulated and inspired me for 12 months as visiting period at the Institute of Computer Graphics and Vision at the Technical University of Graz in Austria.

Thank you to all the crazy guys I met during those years. Thank you to those crazy designers with their moustaches and ties, especially Andrea Gaiardo, Andrea Di Salvo, professor Claudio Germak, professor Paolo Tamborrini and professor Fabrizio Valpreda, I had the pleasure to meet at the beginning of my career. And those during my path Maria Luce Lupetti and Luca Giuliano.

Thank you to all my friends, here in Italy, in Austria and everywhere they are now that loving me continue to support me and motivate me in all my challenges.

Thank you to all the people that helped me with a lot of effort to become what I am today making a perfect plan A, for convincing me then to take plan B.

Everyone has a plan, until they get into the ring and punched in the face (Mike Tyson).

1. INTRODUCTION

In this current period the emerging trend of the Internet of Things, that sees smart objects connected and managed by cloud computing platforms connected over the Internet, plays a key role. Cloud Robotics is an emerging field in robotics that exploits cloud computing, cloud storage and internet technologies for creating robotic services. Cloud Robotics enables robots to benefit from the powerful computational, storage, and communications resources. Moreover it removes overheads for maintenance and updates, and reduces dependence on custom middleware. In this dissertation are presented different technique of Human Robot Interaction with Cloud Robotics applications and services in particular takin into account the emerging technological trends of Unmanned Aerial Vehicles UAV. In this thesis are analyzed the most innovative technique of interactions, starting from the definition of flying robot (UAV) and continuing with the cloud robotics platform that creates new scenarios of service and consequently new ways of interaction.

In the first chapter: there is the introduction part, the definition of UAV and the current ways for interacting with UAVs.

In the second chapter: there is a brief explanation of the architecture of the cloud robotics platform.

In the third chapter: there is the description of the project Fly For Smart City and the related services and interaction techniques.

In the fourth chapter: there is the description of the User’s Flying Organizer (UFO) project

In the fifth chapter: there is the camera to projector calibration problem and solution.

1.1. WHAT IS AN UNMANNED AERIAL VEHICLE?

An Unmanned Aerial Vehicle UAV is defined as an automatic flying device (Nonami, 2010), called colloquially drone.

In general:

- as the name suggests the pilot is not present on board

- flight management is controlled by an automatic system or a remote pilot or from another vehicle

In technical jargon the acronym UA identify an Unmanned Aircraft, indicating a vehicle without on-board pilot. An unmanned vehicle remote controlled is called Remotely Piloted Vehicle RPV, while if it is an aircraft its name changes in remotely piloted aircraft RPA or remotely operated aircraft ROA.

1.2. CLASSIFICATION

During many years of research and development UAV evolved improving their capabilities, endurance, operating altitude and aerodynamics. This brought to a huge amount of category, shapes and sizes. Therefore there are many ways to classify UAVs. In general a wide adopted taxonomy is done according to shape and size configuration (Nonami, 2010). This taxonomy has 4 category:

- Fixed-wing. These Aerial Vehicles AV are the most similar to traditional airplanes. They require a runaway to take-off and land or a catapult launching. They have generally quite long endurance and can fly at high speed (Fig. 1).

- Rotary-wing. Also called rotorcraft UAVs or vertical take-off and landing VTOL, they are similar to helicopters. They have generally high maneuverability and hovering capabilities. Very popular in this category are rotorcraft UAV with 4 propellers called quadrotor. There are platforms with 6, 8 or more propellers and in this case they are called respectively hexacopter, octocopter and so further.

- Blimps such as balloons or airships. They are lighter than air and have long endurance, fly at low speeds, and generally have large size.

- Flapping- wing. They are inspired by birds and flying insects. They have flexible small wings to perform the flight.

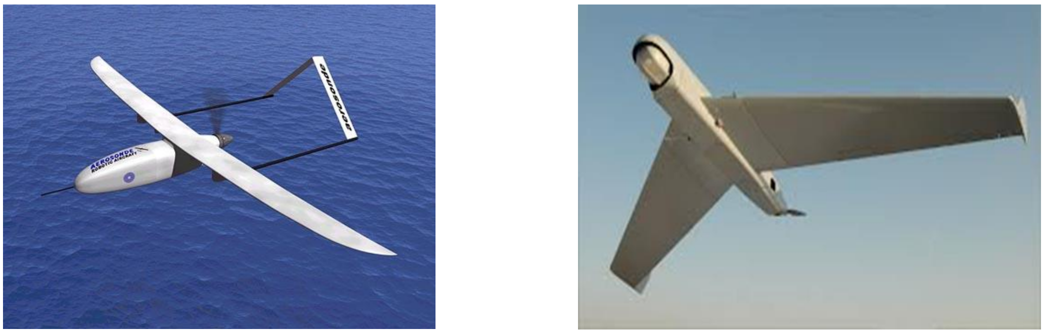


FIGURE 1ON THE LEFT THE AEREO SONDE AND ON THE RIGHT THE BOOMERANG UAV

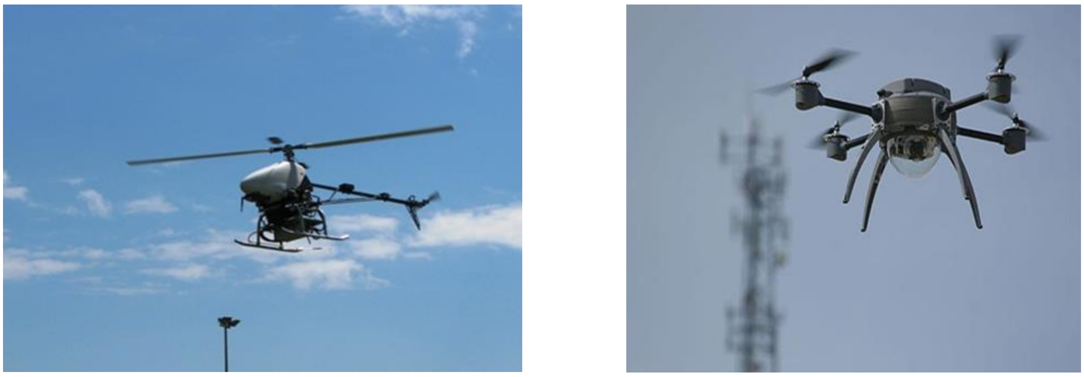


FIGURE 2ON THE LEFT THE ROTARY-WING BUILT BY CSIR AND ON THE RIGHT THE AERY ON SCOUT BY ARERY ON LABS

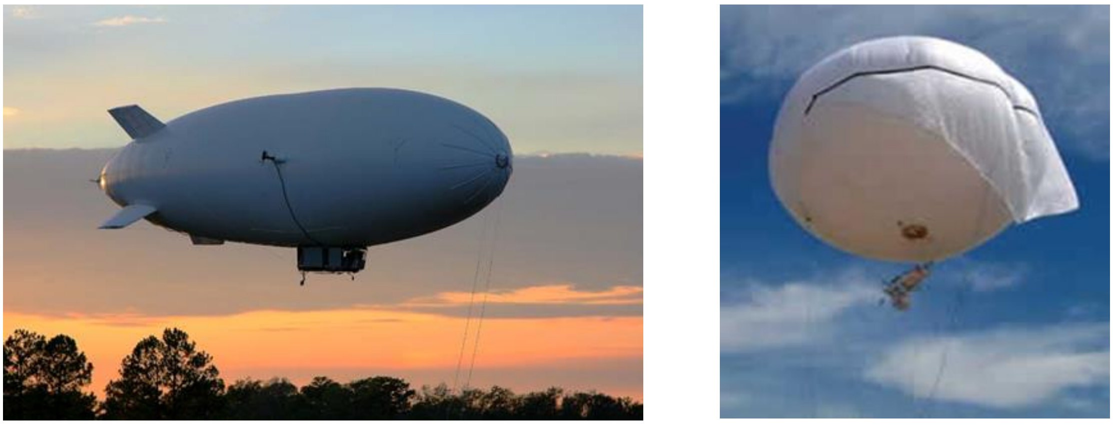


FIGURE 3BLIMPS

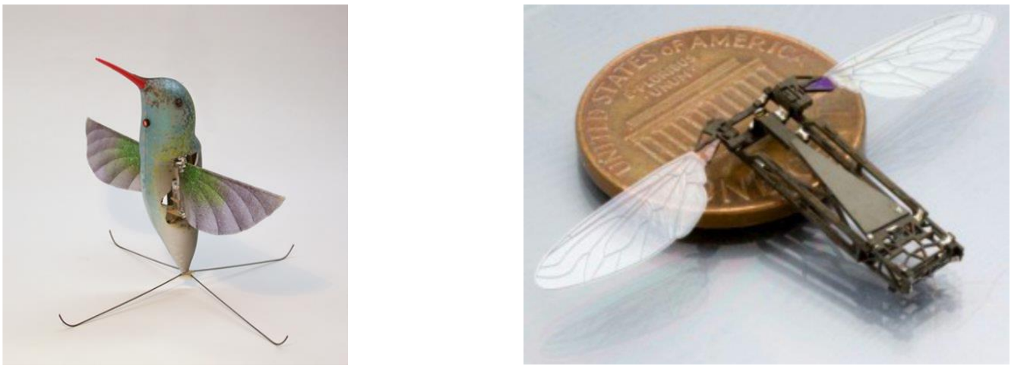


FIGURE 4 ON THE LEFT THE NANO HUMMINGBIRD. ON THE RIGHT THE MAV (MICRO AERIAL VEHICLE) DEVELOPED BY THE HARVARD UNIVERSITY

Another categorization can be done according to the mission they will have to carry out (Austin, 2011):

- HALE high altitude long endurance. Over 15 000 m altitude and 24+ hours endurance. They can carry out extremely long-range (trans-global) reconnaissance and surveillance missions. Generally operated from fixed bases.

- MALE medium altitude long endurance. 5000–15 000 m altitude and 24 hours endurance. Their roles are similar to the HALE systems but generally operate at somewhat shorter ranges, but still in excess of 500 km. and from fixed bases.

- TUAV Medium Range or Tactical UAV with range of order between 100 and 300 km. These air vehicles are smaller and operated within simpler systems than are HALE or MALE

- Close Range-UAV. They usually operate at ranges of up to about 100 km and have probably the most prolific of uses in both fields, including roles as diverse as reconnaissance, target designation, monitoring, airfield security, ship-to-shore surveillance, power-line inspection, crop-spraying and traffic monitoring, etc .

- Mini UAV. They are UAV of below a certain mass (yet to be defined) probably below 20 kg, but not as small as the MAV, capable of being hand-launched and operating at ranges of up to about 30 km.

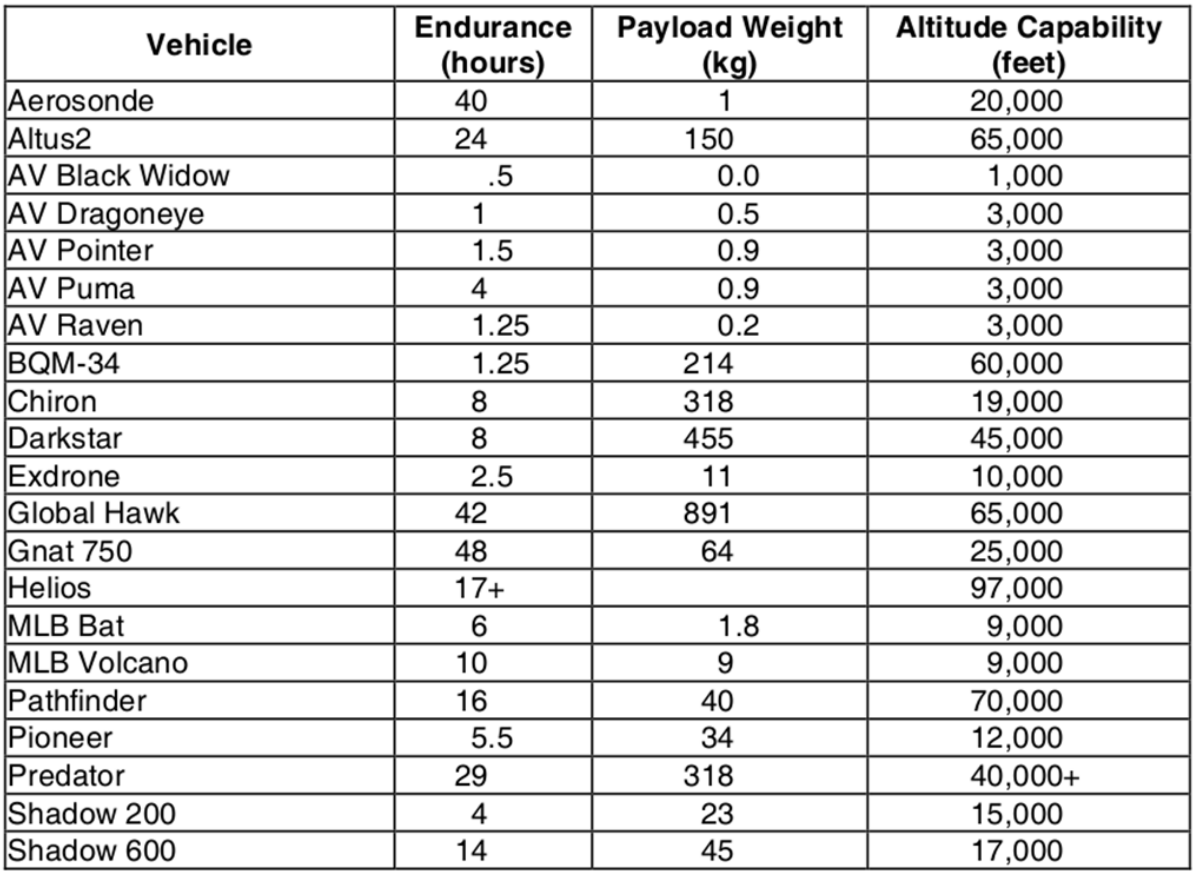
- MAV as originally defined as a UAV having a wing-span no greater than 150 mm. This has now been somewhat relaxed but the MAV is principally required for operations in urban environments, particularly within buildings. It is required to fly slowly, and preferably to hover and to 'perch' – i.e. to be able to stop and to sit on a wall or post. To meet this challenge, research is being conducted into some less conventional configurations such as flapping wing aircraft. Since they present technical problems they didn't find yet applications outside the research labs.

NAV These are proposed to be of the size of sycamore seeds and used in swarms for purposes such as radar confusion or conceivably, if camera, propulsion and control sub-systems can be made small enough, for ultra-short range surveillance.

Important factors for designing a service/applications are:

- payload
- endurance (battery, energy consumption)
- operation area, maximum distance
- maximum velocity
- take-off and landing equipment
- operating altitude
- operating environment

Below there is a comparative table from (Brecher, December 2003.). It is important to notice that in this analysis there are only small and fixed-wing UAV.



Vehicle	Endurance (hours)	Payload Weight (kg)	Altitude Capability (feet)
Aerosonde	40	1	20,000
Altus2	24	150	65,000
AV Black Widow	.5	0.0	1,000
AV Dragoneye	1	0.5	3,000
AV Pointer	1.5	0.9	3,000
AV Puma	4	0.9	3,000
AV Raven	1.25	0.2	3,000
BQM-34	1.25	214	60,000
Chiron	8	318	19,000
Darkstar	8	455	45,000
Exdrone	2.5	11	10,000
Global Hawk	42	891	65,000
Gnat 750	48	64	25,000
Helios	17+		97,000
MLB Bat	6	1.8	9,000
MLB Volcano	10	9	9,000
Pathfinder	16	40	70,000
Pioneer	5.5	34	12,000
Predator	29	318	40,000+
Shadow 200	4	23	15,000
Shadow 600	14	45	17,000

Below a more general analysis from (Peschel).

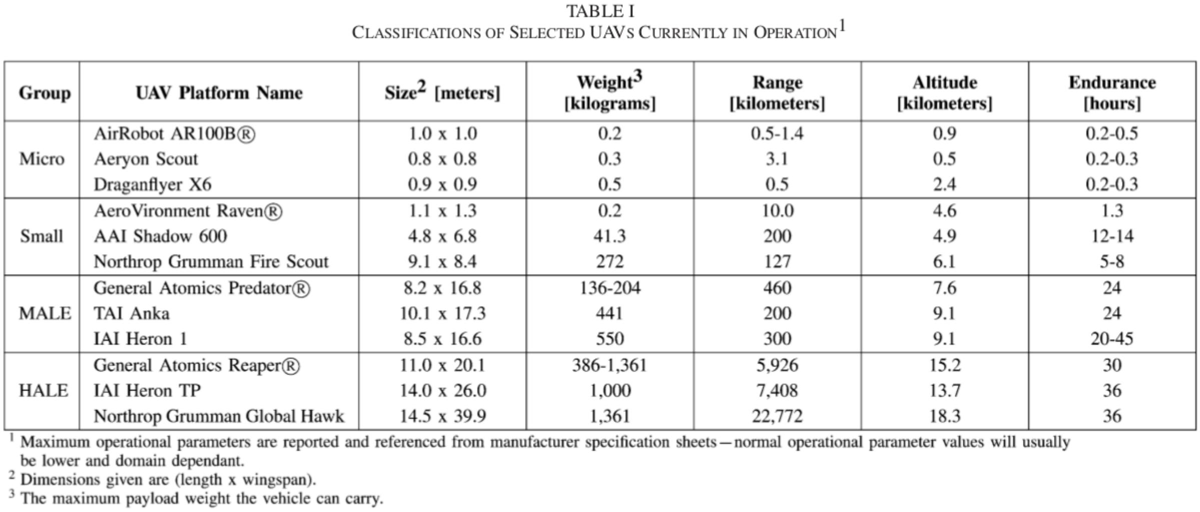


TABLE I

CLASSIFICATIONS OF SELECTED UAVS CURRENTLY IN OPERATION[1]

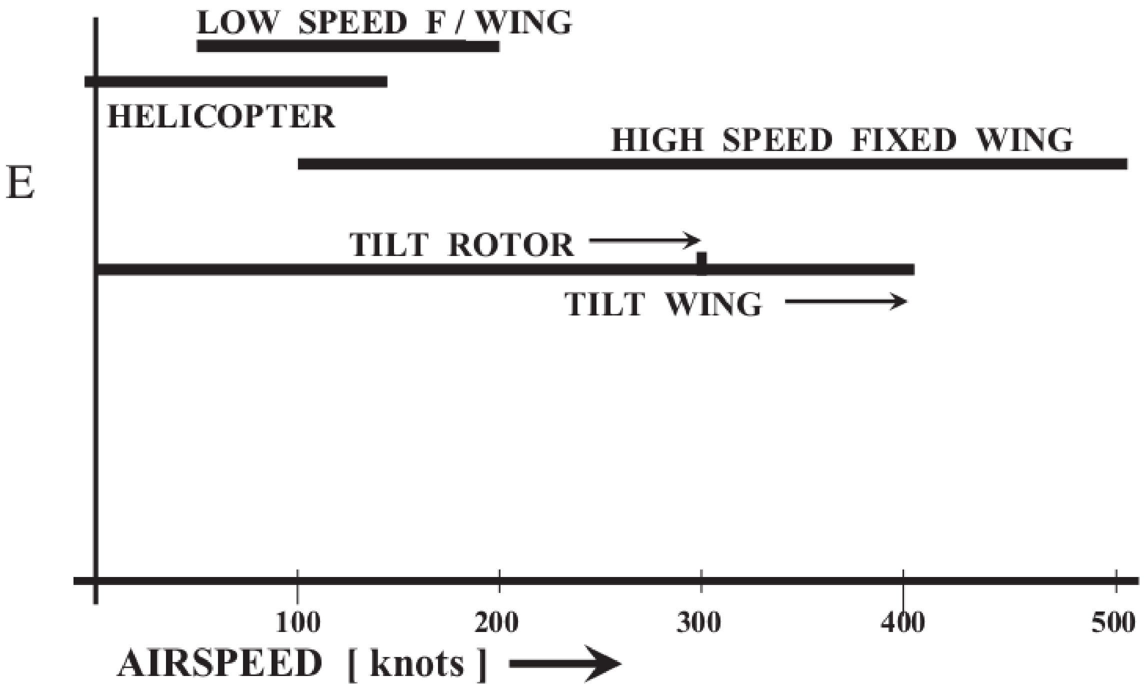
Group	UAV Platform Name	Size[2] [meters]	Weight[3] [kilograms]	Range [kilometers]	Altitude [kilometers]	Endurance [hours]
Micro	AirRobot AR100B®	1.0 x 1.0	0.2	0.5-1.4	0.9	0.2-0.5
	Aeryon Scout	0.8 x 0.8	0.3	3.1	0.5	0.2-0.3
	Draganflyer X6	0.9 x 0.9	0.5	0.5	2.4	0.2-0.3
Small	AeroVironment Raven®	1.1 x 1.3	0.2	10.0	4.6	1.3
	AAI Shadow 600	4.8 x 6.8	41.3	200	4.9	12-14
	Northrop Grumman Fire Scout	9.1 x 8.4	272	127	6.1	5-8
MALE	General Atomics Predator®	8.2 x 16.8	136-204	460	7.6	24
	TAI Anka	10.1 x 17.3	441	200	9.1	24
	IAI Heron 1	8.5 x 16.6	550	300	9.1	20-45
HALE	General Atomics Reaper®	11.0 x 20.1	386-1,361	5,926	15.2	30
	IAI Heron TP	14.0 x 26.0	1,000	7,408	13.7	36
	Northrop Grumman Global Hawk	14.5 x 39.9	1,361	22,772	18.3	36

[1] Maximum operational parameters are reported and referenced from manufacturer specification sheets—normal operational parameter values will usually be lower and domain dependant.

[2] Dimensions given are (length x wingspan).

[3] The maximum payload weight the vehicle can carry.

In the chart below on the x-axis there is the velocity and on the y-axis the efficiency of any topology of UAV (Austin, 2011).



1.3. UAS

So far it has been explained different categories of UAVs. Actually the UAV is part of a much more complex system that has to be correctly designed when developing a service (e.g. search and rescue, reconnaissance, surveillance). This system is called Unmanned Aerial System UAS.

The UAS is composed of (Austin, 2011) :

- Control Station (CS), Usually based on the ground (GCS), or aboard ship (SCS), though possibly airborne in a 'parent' aircraft (ACS), the control station is the control center of the operation and the man–machine interface. It can be the control interface where the mission is preloaded or uploaded in real-time or the UAV teleoperation (Human Machine Interface)
- Payload, is the load that can carry the UAV. It can vary from 200g to 1 Tons.
- Navigation system. Nowadays the navigation system is based on GPS technology. If GPS wasn't applicable there are other technique:
  - radar, UAV position is calculated through a radar in the Control Station
  - radio, the localization is calculated knowing the speed and time that the signal takes to go from the UAV and the CS
  - direct recognition, with the computer-integration of velocity vectors and time elapsed, the aircraft position may be calculated

o Launch and recovery equipment, small UAV can be hand-launched by or with an ad-hoc equipment similar to ramps or rubber bungees

o Communication link, usually the operator manage the flight and/or the mission through an up-link communication. While it is called communication down-link the communication link used to convey information to the CS retrieved by the UAV (e.g. images, video, telemetry).

o Support and transport equipment, the equipment that enable the UAS management and transport

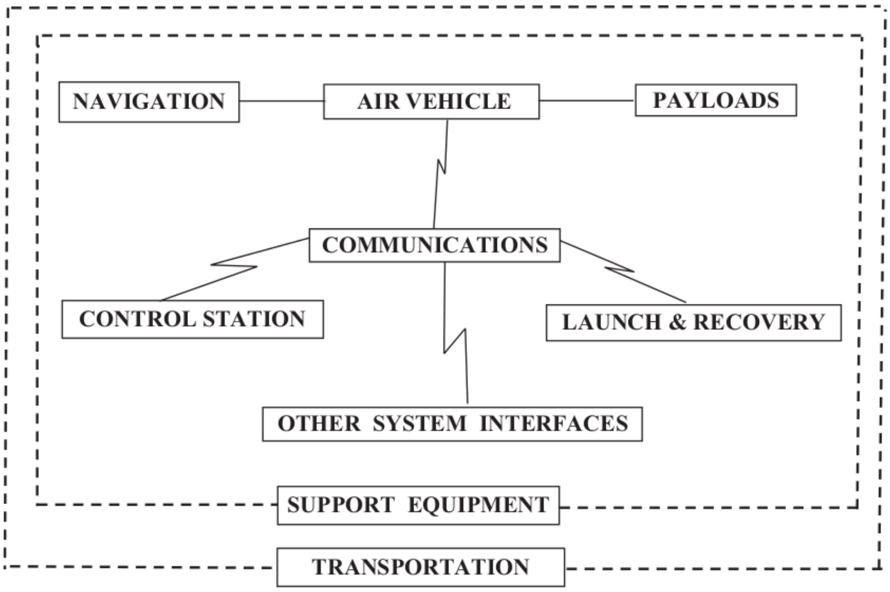


FIGURE 5 UNMANNED AERIAL SYSTEM

1.4. APPROACH

Regarding the interaction between humans and UAVs, we define three different approaches for exploring the state of the art of such HMI:

- Direct interaction with interface: generally this interface can be located in the GS and it represents the mean of communication between user and UAV. The operator through such interface manages the flight, navigation, the payload and information from and to the UAV.

- Direct interaction (without interface): this is the direct interaction with no interface. This is possible for example with gesture, gaze or voice recognition.

- Indirect interaction: in this Cloud Robotics context strictly connected with the Internet of Things usually there is an high level management platform. This platform has to operate, manage UAVs and any devices that is in the environment e.g. sensor networks. If we imagine for example a traffic monitoring system, the platform has to manage UAVs and sensors in the city environment organizing the information and making it accessible to users via e.g. smartphone applications

1.5. DIRECT INTERACTION

In this paragraph there is an overview of the state of the art interfaces for interacting with UAVs. It is important to notice that this kind of interaction is the oldest and the widest adopted in UAV services. In fact, as it is explained in the previous chapter, the man-machine interaction is in the Control Station, often in our applications called Ground Station because it is located on the ground. It may be simply the control center of a local UAV system, within which the mission is pre-planned and executed, but may also be part of a still larger system, sharing information with and receiving information from other elements of the system (Austin, 2011).

The operator sends the flight profile or updates the mission flight via the communication system up-link from the Control Station . The aircraft will return information and images to the operators via the communications down-link. The information will usually include data from the payloads, status information on the aircraft, altitude and airspeed and terrestrial position information.

In addition the CS can communicate with other systems for:

- acquiring weather data,
- transferring information from and to other systems in the network,
- asking services or information from higher authority, and
- reporting information back to that or other authorities.

To give an example, below there is a popular CS for controlling the Desert Hawk III, mini-UAV.

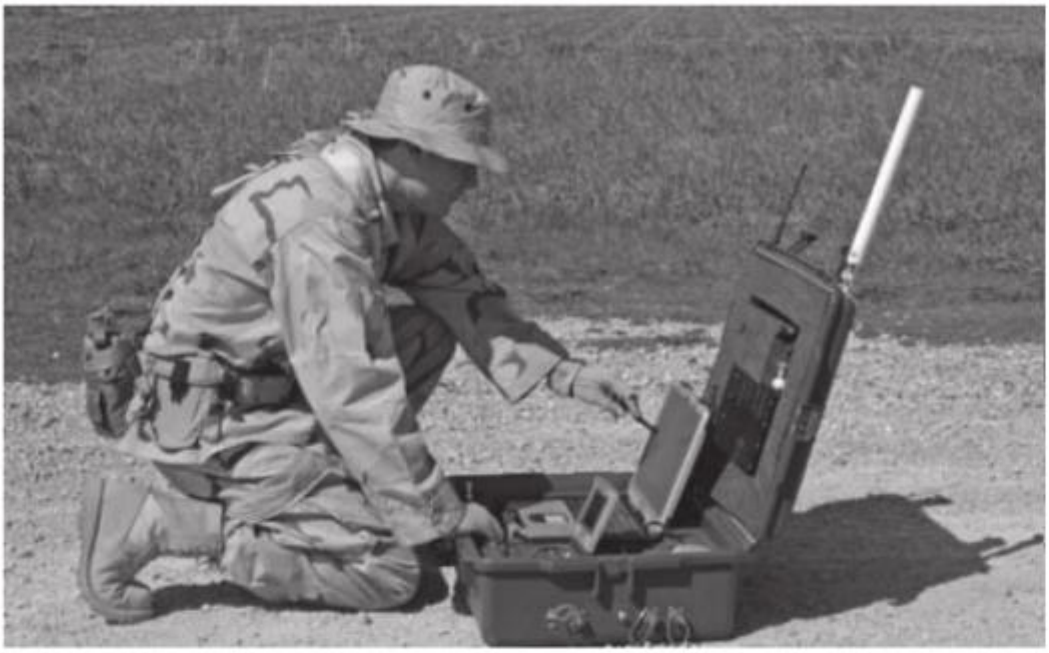


FIGURE 6 DESERT HAWK III GROUND STATION

This was a back-packed control station with communication system and antenna. The GCS incorporates the graphical user interface (GUI) and features a touch-screen laptop, allowing operators easily to input way-points onto a map base. This UAV uses also digital terrain elevation data (DTED) with terrain contours for additional mission assurance .

Control Stations can vary according to UAV types, application and size. Referring to (Peschel) the team that is in charge of operating and managing UAV mission is composed by:

- Flight Director
- Pilot
- Mission Specialist

Flight Director: The role can be held from one or more people. This role is responsible for overall safety of the team members and is in charge of mission situational awareness. The Flight Director has the authority to terminate the mission at any point. As it is from (Peschel) there are similar roles according to other authors such as:

- Incident Commander, who is responsible of the search effort in the micro UAV category
- Team Commander, in the small UAV category is the head of the Human Robot team, he/she can communicate with other small UAV and monitor the condition of the vehicle
- Mission Commander, is an additional individual who has to deal solely with strategy and coordination

Pilot: one or more people responsible for flight control activities. For the micro UAV category the pilot role involves also aviation, navigation and maintenance of the vehicle. In general when the UAV is autonomous the pilot role is moved in an offsite control center. In this condition this role decides where the vehicle should fly or monitors flight and data collection.

Mission specialist: is the team member responsible for visual investigation and recording or in more advanced vehicle system, delivery of an on-board payload.

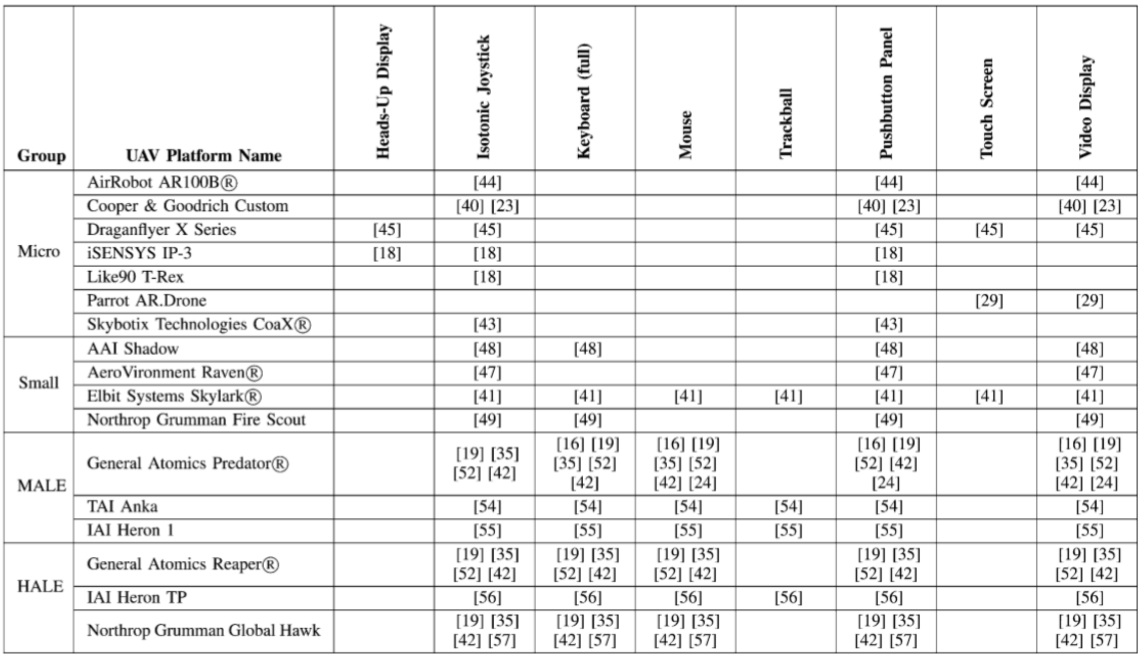
Regarding the interfaces, they can change from application to application depending on the degree of autonomy of the UAV. In fact in literature there is a distinction between Exocentric and Egocentric Viewpoint .The control station can either be designed to make the pilot feel as though the operational perspective is within the aircraft looking outside (egocentric) or outside the aircraft watching it from afar (exocentric) (Williams, 2007).

There is a typical Human Computer Interaction open-issue that is the balance between an autonomous, semi-autonomous or manual operation of UAV or in more general a robot. For the focus of this report more related to UAV interaction a value work is (Murphy, 2008).

There are also guidelines for designing effective Human-Machine Interfaces such as ARINC (Radio.), STANG 4586 (Cummings et al., 2006), (NATO: The North Atlantic Treaty Organization, 2007), DO-178B (RTCA: Radio Technical Commission for Aeronautics, 1992), JAUS (Society of Automotive Engineers SAE, 2010), ISO 9241-11 (Standardization, 2000) and GEDIS (Ponsa, 2007).

According to R. Murphy (Peschel) the pilot role is a well-known problem in HCI that is also analyzed in other field e.g. automotive industry, remote vehicles. An interesting class of interfaces from (Peschel) is Human-Machine Systems of the Mission Specialist. Below there is the reference table of summary of hardware-based HMI used by a mission specialist.

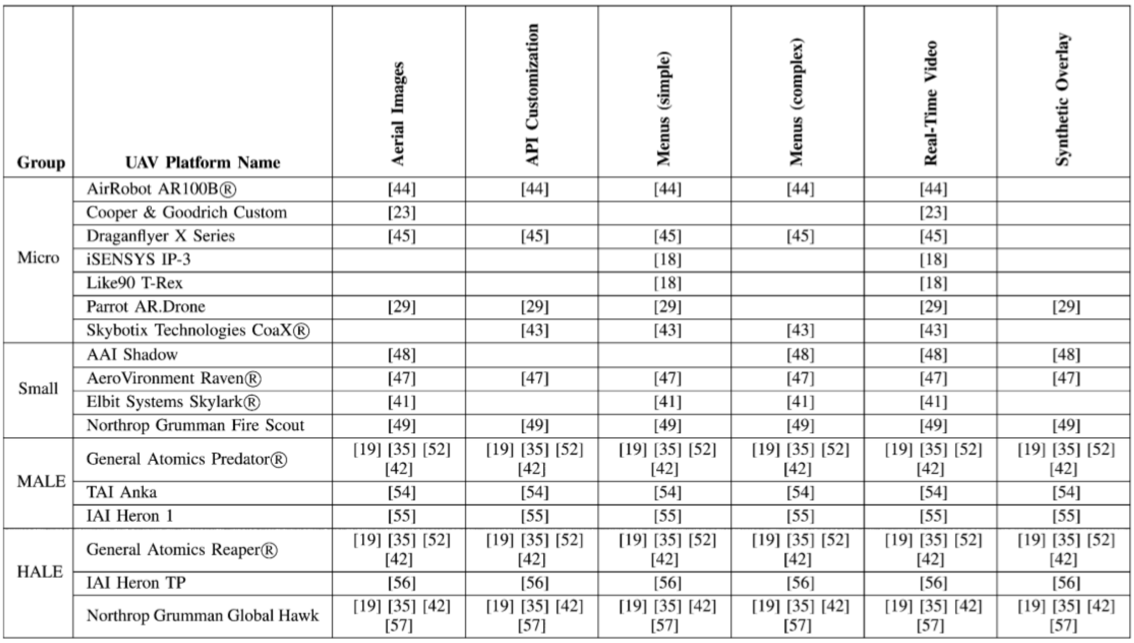
TABLE III
REFERENCE SUMMARY OF HARDWARE-BASED HMI USED BY A MISSION SPECIALIST ROLE ON A UAS HUMAN–ROBOT TEAM



Group	UAV Platform Name	Heads-Up Display	Isotonic Joystick	Keyboard (full)	Mouse	Trackball	Pushbutton Panel	Touch Screen	Video Display
Micro	AirRobot AR100B®		[44]				[44]		[44]
	Cooper & Goodrich Custom		[40] [23]				[40] [23]		[40] [23]
	Draganflyer X Series	[45]	[45]				[45]	[45]	[45]
	iSENSYS IP-3	[18]	[18]				[18]		
	Like90 T-Rex		[18]				[18]		
	Parrot AR.Drone							[29]	[29]
	Skybotix Technologies CoaX®		[43]				[43]		
Small	AAI Shadow		[48]	[48]			[48]		[48]
	AeroVironment Raven®		[47]				[47]		[47]
	Elbit Systems Skylark®		[41]	[41]	[41]	[41]	[41]	[41]	[41]
	Northrop Grumman Fire Scout		[49]	[49]			[49]		[49]
MALE	General Atomics Predator®		[19] [35] [52] [42]	[16] [19] [35] [52] [42]	[16] [19] [35] [52] [42] [24]		[16] [19] [52] [42] [24]		[16] [19] [35] [52] [42] [24]
	TAI Anka		[54]	[54]	[54]	[54]	[54]		[54]
	IAI Heron 1		[55]	[55]	[55]	[55]	[55]		[55]
HALE	General Atomics Reaper®		[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]		[19] [35] [52] [42]		[19] [35] [52] [42]
	IAI Heron TP		[56]	[56]	[56]	[56]	[56]		[56]
	Northrop Grumman Global Hawk		[19] [35] [42] [57]	[19] [35] [42] [57]	[19] [35] [42] [57]		[19] [35] [42] [57]		[19] [35] [42] [57]

The table below is the reference summary of software-based HMI for the mission specialist team member.

TABLE IV
REFERENCE SUMMARY OF SOFTWARE-BASED HMI USED BY A MISSION SPECIALIST ROLE ON A UAS HUMAN–ROBOT TEAM



Group	UAV Platform Name	Aerial Images	API Customization	Menus (simple)	Menus (complex)	Real-Time Video	Synthetic Overlay
Micro	AirRobot AR100B®	[44]	[44]	[44]	[44]	[44]	
	Cooper & Goodrich Custom	[23]				[23]	
	Draganflyer X Series	[45]	[45]	[45]	[45]	[45]	
	iSENSYS IP-3			[18]		[18]	
	Like90 T-Rex			[18]		[18]	
	Parrot AR.Drone	[29]	[29]	[29]		[29]	[29]
	Skybotix Technologies CoaX®		[43]	[43]	[43]	[43]	
Small	AAI Shadow	[48]			[48]	[48]	[48]
	AeroVironment Raven®	[47]	[47]	[47]	[47]	[47]	[47]
	Elbit Systems Skylark®	[41]		[41]	[41]	[41]	
	Northrop Grumman Fire Scout	[49]	[49]	[49]	[49]	[49]	[49]
MALE	General Atomics Predator®	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]
	TAI Anka	[54]	[54]	[54]	[54]	[54]	[54]
	IAI Heron 1	[55]	[55]	[55]	[55]	[55]	[55]
HALE	General Atomics Reaper®	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]	[19] [35] [52] [42]
	IAI Heron TP	[56]	[56]	[56]	[56]	[56]	[56]
	Northrop Grumman Global Hawk	[19] [35] [42] [57]	[19] [35] [42] [57]	[19] [35] [42] [57]	[19] [35] [42] [57]	[19] [35] [42] [57]	[19] [35] [42] [57]

Regarding the direct interaction between human and drone there are many research projects that are analyzing this interaction in particular with gestures. A value work was carried out in (Guo, 2008.) comparing different gestures metaphors. In this work has been developed 3D interaction techniques using the Microsoft Kinect SDK to control the AR Drone 2.0 .

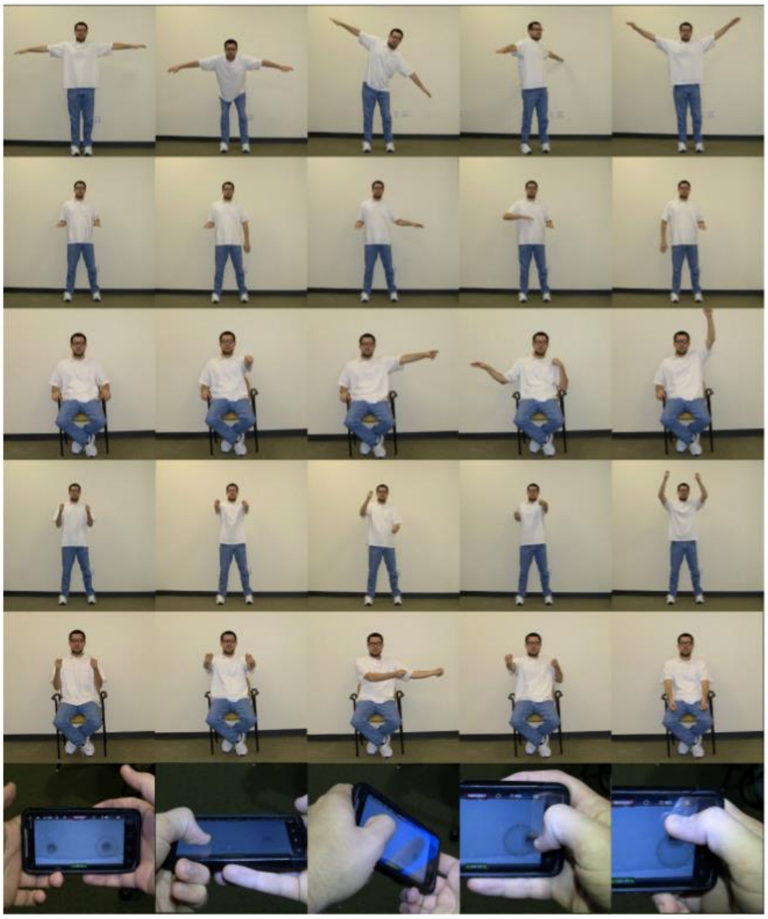


FIGURE 7 3D GESTURE METAPHORS

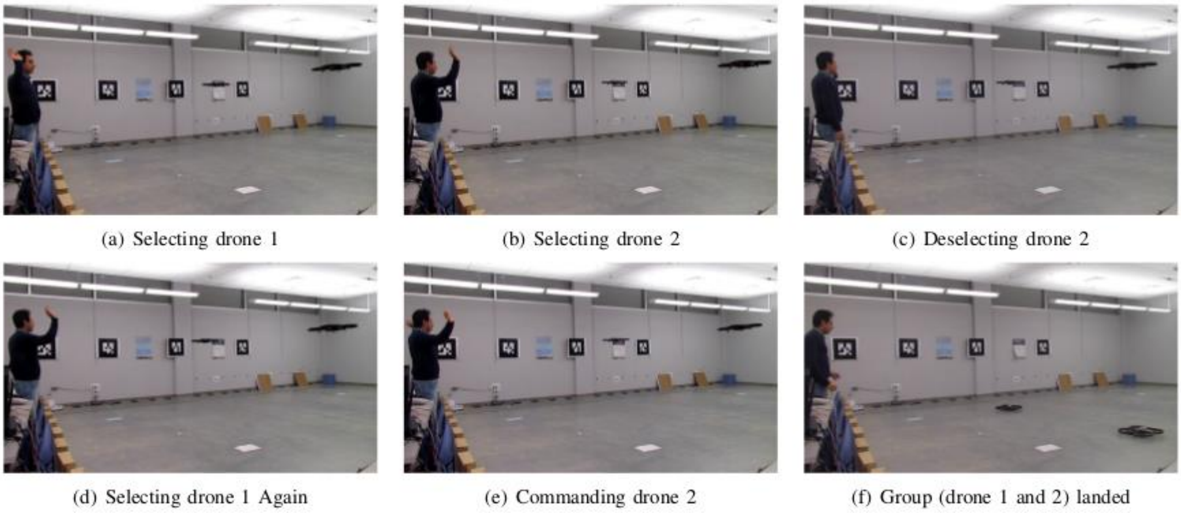
Five 3D gestural interfaces, were selected and compared to each other and to a smartphone interface. Techniques include a first-person interaction metaphor where a user takes a pose like a winged aircraft, a game controller metaphor, where a user's hands mimic the control movements of console joysticks, "proxy" manipulation, where the user imagines manipulating the UAV as if it were in their grasp, and a pointing metaphor in which the user assumes the identity of a monarch and commands the UAV as such.

As final result two of these gestures were considered the best due to varying results in the factors of comfort, likeability, naturalness and overall perception:

- Proxy. This technique requires both arms in tandem to perform any command (second gesture from the top in the picture). It is considered as the best suitable technique for non-recreational use.

- First person, This technique is based on the metaphor of the user assuming the pretense role of an aircraft. This is probably more suitable for entertainment purposes (first from the top).

While regarding the interaction with a team of UAV a value work is (Monajjemi, 2013). In this work the user selects an individual as the current focus of attention by simply looking at it. The focused robot can be added/removed from the team by waving the right/left hand. The whole team is dispatched to a mission by waving both hands.


(a) Selecting drone 1 (b) Selecting drone 2 (c) Deselecting drone 2

(d) Selecting drone 1 Again (e) Commanding drone 2 (f) Group (drone 1 and 2) landed

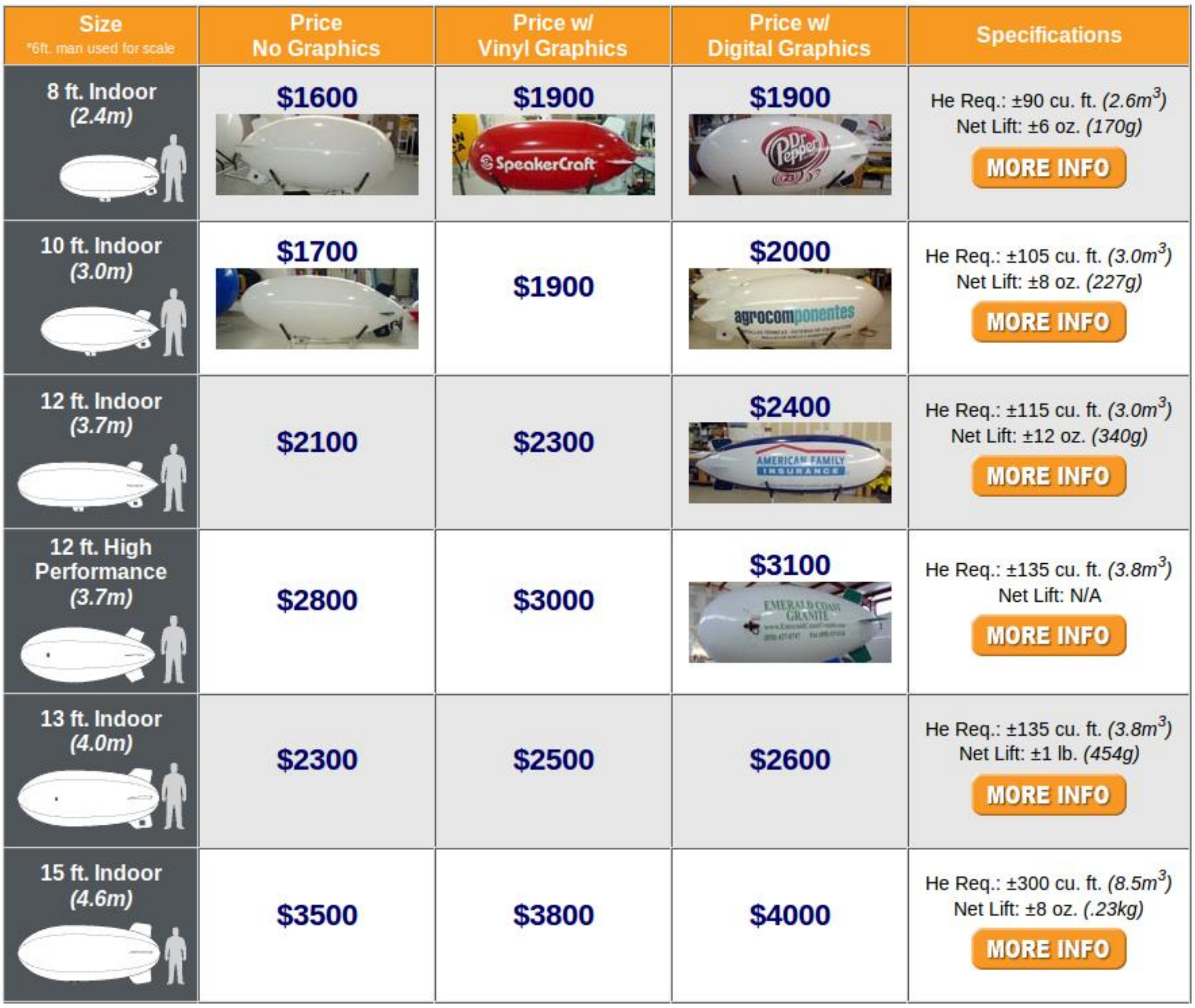
FIGURE 8 HRI IN THE SKY

Although the works so far analyze the direct interaction with quadrotors many problems can come out applying this technology especially in indoor environment:

- noise deriving from the propellers,
- unreliable/unstable control,
- constrains on battery duration,
- unsafely human-machine interaction.

Obviously blimps compared to quadrotors present a payload restricted according to their volume, the available battery power, the computational resources, the sensing capabilities of the employed sensors, and also the rotor power are limited. But according to (Liew, 2013) blimps seem more suitable for a direct human-machine interaction in indoor environment. Interesting experiments were carried out in (Burri) and in (Müller, 2013).

Regarding the ratio payload and size we have (http://www.southernballoonworks.com/rc-blimps/remote-control-blimps.html):



Size *6ft. man used for scale	Price No Graphics	Price w/ Vinyl Graphics	Price w/ Digital Graphics	Specifications
8 ft. Indoor *(2.4m)*	$1600	$1900	$1900	He Req.: ±90 cu. ft. *(2.6m^3)* Net Lift: ±6 oz. *(170g)* MORE INFO
10 ft. Indoor *(3.0m)*	$1700	$1900	$2000	He Req.: ±105 cu. ft. *(3.0m^3)* Net Lift: ±8 oz. *(227g)* MORE INFO
12 ft. Indoor *(3.7m)*	$2100	$2300	$2400	He Req.: ±115 cu. ft. *(3.0m^3)* Net Lift: ±12 oz. *(340g)* MORE INFO
12 ft. High Performance *(3.7m)*	$2800	$3000	$3100	He Req.: ±135 cu. ft. *(3.8m^3)* Net Lift: N/A MORE INFO
13 ft. Indoor *(4.0m)*	$2300	$2500	$2600	He Req.: ±135 cu. ft. *(3.8m^3)* Net Lift: ±1 lb. *(454g)* MORE INFO
15 ft. Indoor *(4.6m)*	$3500	$3800	$4000	He Req.: ±300 cu. ft. *(8.5m^3)* Net Lift: ±8 oz. *(.23kg)* MORE INFO

FIGURE 9 BLIMPS PAYLOAD AND SIZE

1.6. Indirect Interaction

As explained previously this kind of interaction is the most related to cloud robotics, since it requires an high level platform that manages UAVs and devices in the environment providing services to users. Since the field is relative young (one of the first work was in 2010 (Chen, 2010.)) there is not so much work related, especially from the HRI perspective. The typical architecture is composed of:

– The management platform. It has to manage robots and devices in the environment and provide services accessible to users.

– Services. In this layer HMI services can be located They interact with users and can send or get information to or from the cloud platform.

– Robots. They can operate in the environment and collect data.

The Cloud Robotics is strictly connected to other emerging research themes such as Internet of Things, Cloud Computing, Cyber Physical Systems and Big Data as Ken Glodberg points out in (Goldberg, 2013).

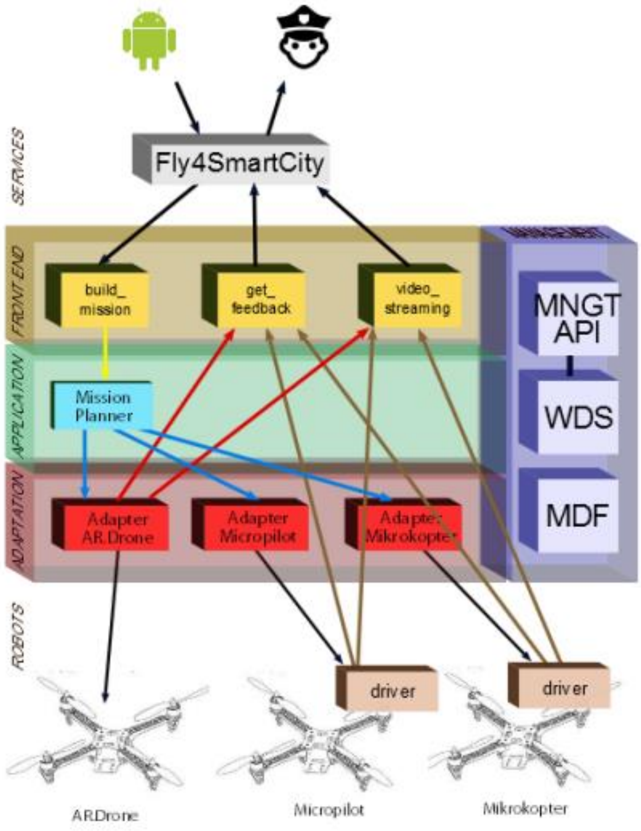


FIGURE 10 A CLOUD ROBOTICS ARCHITECTURE FROM (ERMACORA, 2013)

Obviously the Cloud Robotics has a value potential:

- for increasing the computational capabilities of robots by relying on remote servers for most of their computational load
- for sharing knowledge with other robots
- for abstracting the hardware platforms

But in terms of services/applications robots can play a role of data-collectors, data-sharers and agents in the environment interacting with devices e.g. with sensors, other robots, NFC/RFID.

As Michahelles Florian states in (F. Michahelles, August, 2012) robots are part of the IoT network. Regarding the integration of robots and big data, some researchers are also thinking to apply underwater gliders to predict dangerous storms (http://www.popsci.com/article/technology/underwater-gliders-gather-data-help-predict-next-big-storm) . A much more visionary perspective is in (http://smartdatacollective.com/bernardmarr/109391/big-data-robots-are-they-after-your-job) . Sharing and collecting information can also have legal and privacy issue that is analyzed in (Pagallo, 2013).

Therefore regarding HMI aspects the interfaces will probably change. The evolution of the direct interaction with interface could continue to exist as a cloud services, e.g. the teleoperation interface accessible through a website or a smartphone app. But in the perspective that sees robots as data-collectors or data-sharers, users that access to the cloud robotics service could ignore the existence of robots that provide such data. In this case the HMI is related to the service, and become a typical HCI issue such as interacting with a common web page or a smartphone app.

Obviously the interaction can be also direct even in cloud robotics as it is in (Kanda, 2009). In this work there are a series of abstraction techniques for people's trajectories and a service framework for using these techniques in a social robot, which enables a designer to make the robot proactively approach customers by only providing information about target local behavior. They placed a ubiquitous sensor network consisting of six laser range finders in a shopping arcade. The system tracks people's positions as well as their local behaviors, such as fast walking, idle walking, wandering, or stopping.

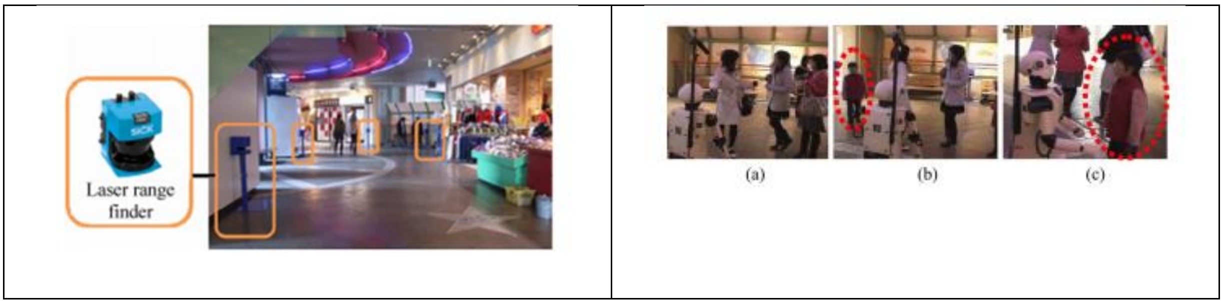


FIGURE 11 FROM (KANDA, 2009) ON THE LEFT LASER SCANNER IN THE ENVIRONMENT. ON THE RIGHT ROBOTS APPROACHING CUSTOMERS

2. The Cloud Robotics Platform

The cloud robotics platform used in this thesis has been designed by Telecom Italia in order to be resilient. Moreover the platform has been designed to:

1) abstract the hardware and software layer to final developer
2) offload demanding data processing and computing
3) expose Application Programming Interfaces (APIs) to ease the development of services and share resources amongst different services

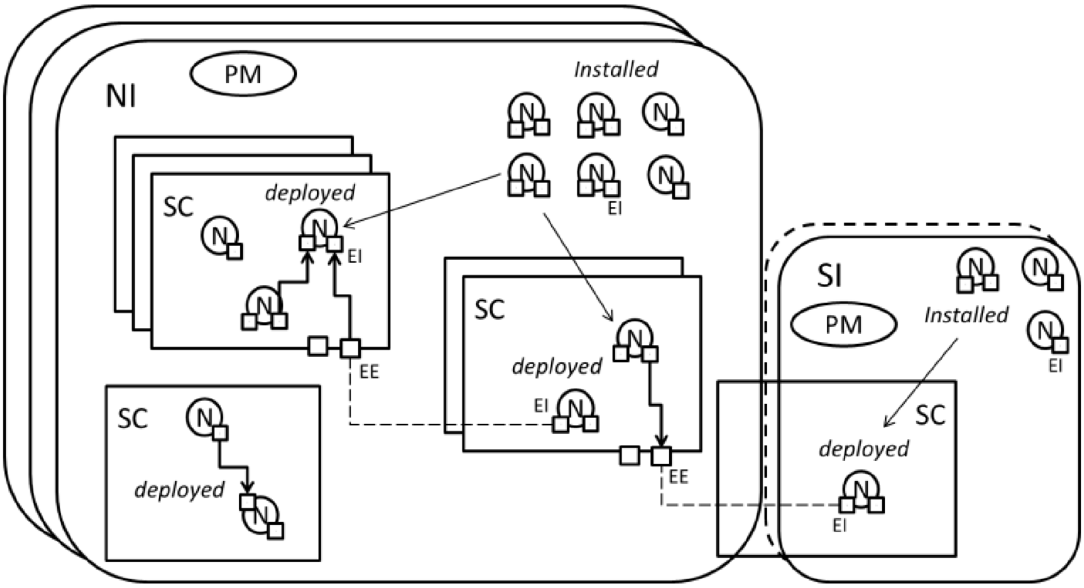


FIGURE 12 CLOUD ROBOTICS PLATFORM: THE PLATFORM OBJECTS AND THEIR RELATIONSHIPS.

In this section we present the basic elements composing the cloud robotics platform (Figure 12):

1) **Node (N)**: is the basic “building block” of a service. It is *installed* if it is within an *instance*. It is *started* if it is within a service container.

2) **Endpoint**: could be:

a. **Internal (IE):** belongs to a node. The IE connects it to another node, in the current service container, or to an external endpoint (EE).

b. **External** (**EE**): belongs to a service container. The EE connects the service container to a node that resides in a different service container. The EE makes it possible to connect nodes belonging to different service containers.

3) **Service Container** (SC): organizes groups of connected (*started*) nodes. Groups of connected nodes are called services.

4) **Instance**: this object contains the platform manager (PM), service containers, nodes and endpoints. It can be:

   a. **Normal (NI)**: if it contains Service Containers and (*installed*) nodes.

   b. **Simple (SI)**: if it only contains (*installed*) nodes and not service containers.

The objects that we described above represent the basis on which robotics applications are built. Applications are based on nodes (N), which are *installed* in instances (SI, NI), or deployed in service containers (SC). *Service APIs* expose platform services while *Management APIs* are used to build a new service.

As a metaphor of the brain, we can say that the network of service containers represents the "remote intelligence"; nodes are the "neurons" ; endpoints are the "synapsis" and the APIs represent the "senses" that enable the interaction with the external world.

2.1. PLATFORM MANAGER

The *Platform Manager* is the object in charge of:

1) sending and receiving commands through the *Command Manager* object;

2) listening and creating events through the *Event Manager*.

Events and commands are accessed by the management API, through *Platform API Manager* element.

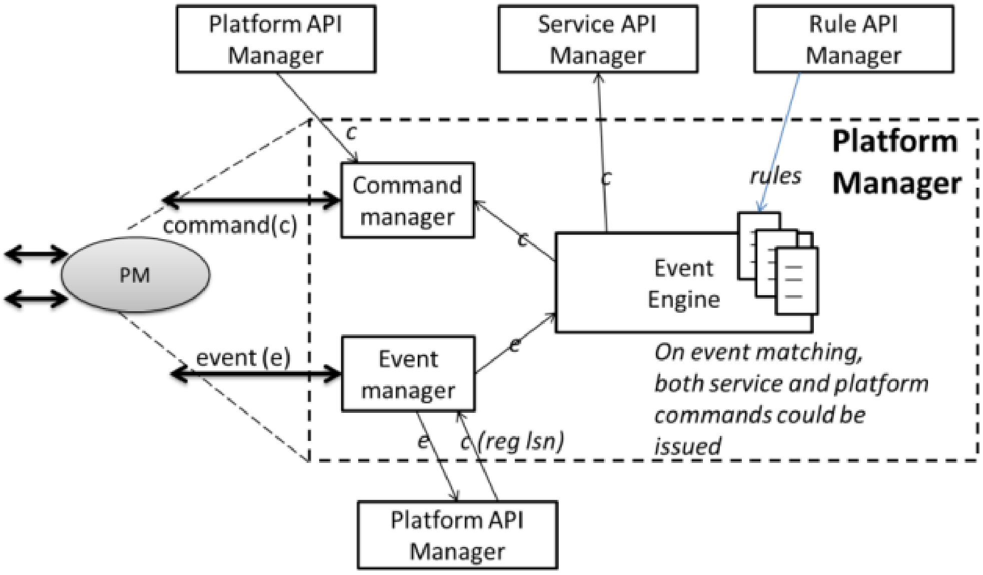
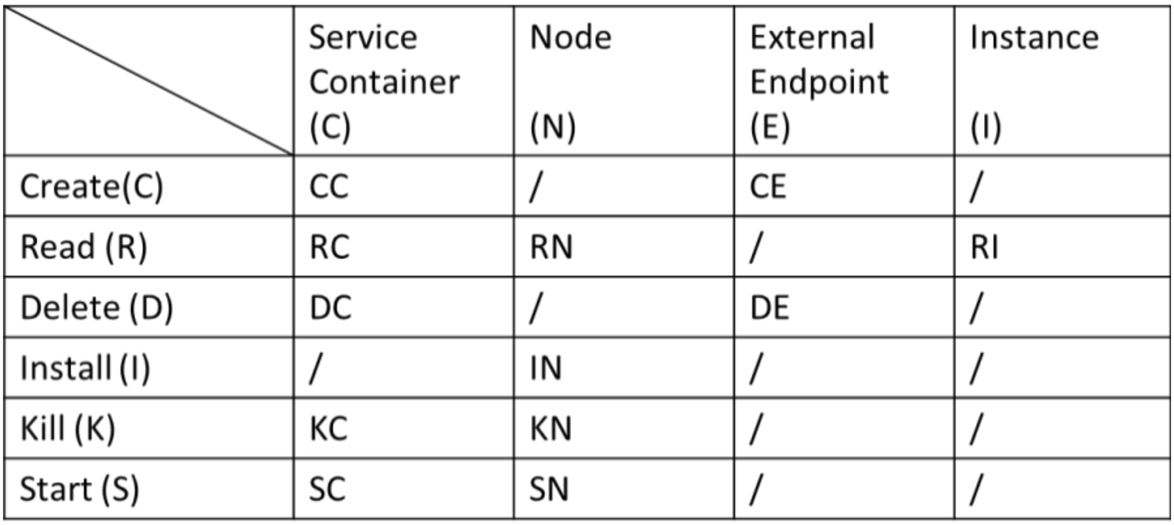


FIGURE 13 PLATFORM MANAGER LOGIC ARCHITECTURE.

The *Event Engine* has been designed to make a platform service more robust and resilient using counteractions. Counteractions are controlled sequence of commands triggered by the matching of pre-configured class of events. The counteractions can be deleted, created and read from applications and users throughout *Rule API Manager*. Commands can be both *platform commands* (e.g., create a service container) or *service commands* (e.g., publish a message). Platform commands are accessible by *Service API Manager* and service commands by *Service API Manager.*

2.2. MANAGEMENTS COMMANDS

The management commands and their structure are listed in Figure x:



	Service Container (C)	Node (N)	External Endpoint (E)	Instance (I)
Create(C)	CC	/	CE	/
Read (R)	RC	RN	/	RI
Delete (D)	DC	/	DE	/
Install (I)	/	IN	/	/
Kill (K)	KC	KN	/	/
Start (S)	SC	SN	/	/

FIGURE 14 PLATFORM MANAGEMENT COMMANDS.

Columns contain the platform objects, while rows contain the possible commands:

1) **Create**: creates an object in the specified instance, if omitted the current instance is the default.
2) **Read**: reads an object and all contained object, i.e. RN command reads the node and all "contained" internal endpoints.
3) **Delete**: deletes an object and all contained objects, i.e. DC command deletes the container, stops all started nodes in it and deletes all "contained" external endpoints.
4) **Install**: installs a node (or more) in current or remote instance.
5) **Kill**: stops a single node (KN) or a group of nodes contained in a service container (KC).
6) **Start**: as for Kill command, it starts both single node (SN) and a group of nodes (SC).

For instance, this is the command for creating a service container named "foo" in an instance name "vm1":

CC I=vm1 foo

As described previously, commands are managed by the Command Manager.

Platform events

Events are classified as follows:

1) **Object (EO)**: object events are raised when an object is created or deleted.

2) **Node (EN):** node events are raised when a node is started, stopped or an error occurs due to unexpected events

3) **Instance (EI):** instance events are raised when an instance is connected or disconnected to the network.

So for example, if an external endpoint named "/bar" is created in a service container named "foo" belonging to a normal instance named "vm1", an EO type event is raised and the related listener can notify the user or the robotics application with the following info:

EO EE vm1.foo./bar created

In this example we can recognize an event type (EO), an object type (EE) and an object name "vm1.foo./bar". The object name has a path reporting the container objects (vm1.foo) and the current status "created".

This event can be received by the event engine and, as a consequence, triggering a sequence of commands, which in turns can raise other events.

2.3. SERVICE AND COMMAND EVENTS

Service commands and events are messages transmitted through connection between two endpoints. The communication paradigm is depicted in figure 4.

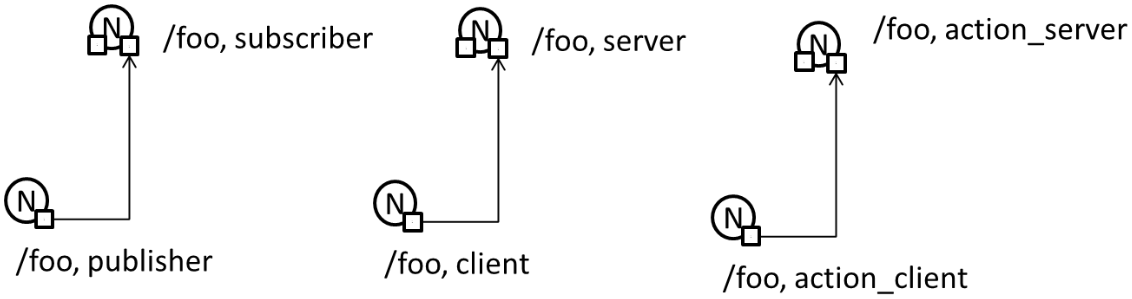


FIGURE 15 COMMUNICATION PARADIGM

Three paradigm of communication are used and reflect the paradigms provided by ROS:

1) publisher-subscriber: a node publishes a message with a publisher endpoint named "/foo". Any node can subscribe the endpoint "/foo" and get the message.

2) client-server: a node sends a request message to a server endpoint “/foo-server” through a client endpoint named “/foo-client”. Another node through the server endpoint “/foo-client” will reply with the response message.

3) action: a node sends a goal message to an action-server endpoint “/foo-action-server” through an action server-client endpoint “/foo-action-client”. Another node will reply with a continuous feedback message on the goal status through “/foo-action-server”. In the end it will send the final message reporting success or error upon reaching the goal.

Each communication process refers to endpoints and not to nodes. In order to let robotic applications and users to communicate through endpoints, the service container exposes endpoints through the Service API Manager, a bridge node to the external world. Let’s suppose that a node resides inside a service container named “roomba”. This node commands the robot wheels, and has a subscriber endpoint named “/vel”. The message type is a structure containing two arrays, the first for linear velocity and the second for the angular one. If an external element needs to stop the robot, the bridge node of the service container named “roomba” has to be connected. The synopsis could be this:

bridge = SAPI.connect(“roomba”)

bridge.publish(“/vel”,[{0 0 0},{0 0 0}])

where **bridge** is the handler of the Service API (SAPI), to address the publisher endpoint that sends message to the service container “roomba”.

2.4. EVENT ENGINE

As outlined before, in the event engine it is possible to create counteractions according to raised events. Counteractions can be composed both by service and platform commands. For example, let’s imagine a WIFI-connected simple instance on a mobile robot that experiences disconnection from platform for WIFI issues. It’s possible to stop the robot by publishing the right service command. Here is the possible synopsis to describe the procedure:

Event e;

e.match(EI robot disconnected):

bridge.publish(/vel,[0 0 0 0 0 0])

where **e** is the internal event engine handler. In the match function of **e**, when the event of the described type occurs the single service command publish is executed. In this case the robot is stopped. When describing event matching and consequent procedures it’s necessary to take into account the following issues:

- As commands indirectly generate events, we need to pay attention to possible undesired loop.

- Matched events can occur with a high frequency. So, it is also important to define time constraints, i.e. matching only once, matching again after a defined number of seconds and so on.

2.5. API MANAGERS

APIs provide access to the platform to external elements (e.g. users, robotics applications). They are divided in (Figure 3):

- *Service API Manager*, this is a node that must be *started* in the service container. Exposing APIs to the external world it makes possible to manage service commands and events.
- *Platform API Manager*. Exposing APIs to the external world it makes possible to manage platform commands and events
- *Rule API Manager*. Exposing APIs to the external world it makes possible to manage the counteractions for the event engine.

3. Fly for Smart City

Cloud computing (Mell, 2011) is revolutionizing the robotics industry, opening new scenarios that embrace the paradigm of the Internet of Things (Atzori, 2010) . This is leading to a paradigm shift where robots now become simple agents that belong to a cloud computing platform (Waibel). The platform manages the robots and provides them with the possibility of sharing knowledge and performing demanding real-time data processing (Chen, 2010.), (Waibel).

In (Sanfeliu) the authors introduced a definition of Network Robot Systems (NRS) as is understood in Europe, USA and Japan.

NRS are defined as any robotic device connected to a communications network which involves hardware, autonomous capabilities, network-based cooperation, and human-robot interaction capabilities. Robot-as-a-Service (RaaS) extends the concept of Service-Oriented Architecture (SOA) to robots, enforcing the design of robotics applications as an all-in-one unit, including services and applications (Chen, 2010.).

In (Quintas, 2011) the authors proposed an approach for an automated system based on a service oriented architecture.

In (Kamei) the requirements in typical daily supporting services have been examined through example scenarios that target senior citizens and the disabled; discussing the key research challenges offered by the cloud network-robotics approach. Similarly, in (Chibani, 2013) the use of cloud robotics has been investigated for physical and virtual companions assisting people in their daily living, ubiquitous robots that are able to co-work alongside people.

In (Waibel) is presented the design and implementation of Rapyuta, an open source Platform-as-a-Service (PaaS) framework for robotics applications.

In this chapter the details about Fly4SmartCity project and vision.

Fly4SmartCity is a large-scale cloud-robotics service that aims to prove that a real world application of cutting-edge technologies in robot-networking and unmanned vehicles is actually possible.

The goal of Fly4SmartCity is to provide a service for emergency management in smart city environment; this is achieved by the strict integration of:

- A capable, reliable and stable cloud platform;
- Sufficient agile autonomous agents;
- A high-bandwidth, low-latency mobile network;
- A rich set of data, drawn from various providers;
- A well-documented and supported robotics middleware.

For validating the service we imagined a test case as follows:

- a citizen is in a dangerous situation and requests the emergency service; using a smartphone application he/she sends back to the cloud platform his/her GPS coordinates and an unique identification number.

- the cloud platform is able to find the most suitable UAV for the mission and sends it to the citizen in order to provide monitoring and support.

- A police officer can access the service in real-time to monitor the actual position of the UAV, its telemetry, and to visualize on the web interface the video stream coming from the UAV's camera.

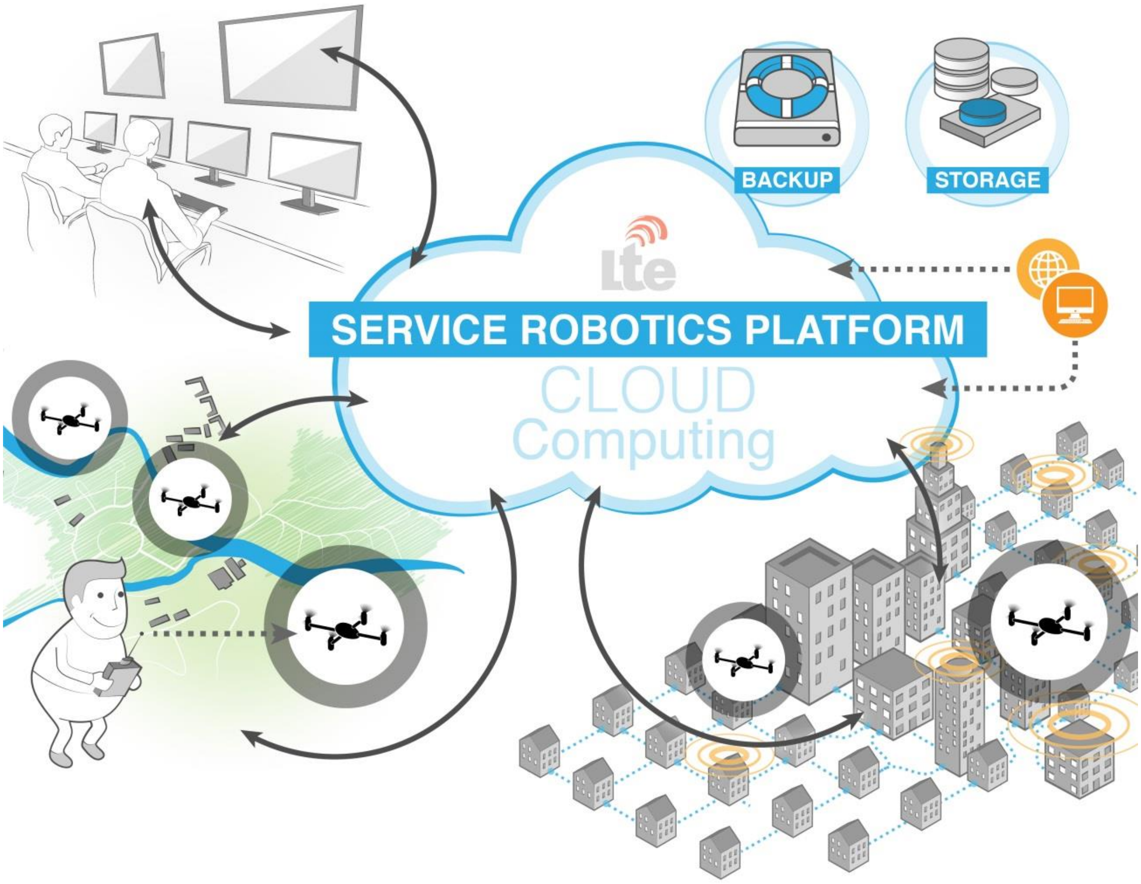


FIGURE 16 THE CLOUD ROBOTICS PLATFORM SUPPORTING DRONE SERVICES

In the Fly*4*SmartCity service there are different applications as urban monitoring,

emergency-management, goods delivery and user-defined flight plan. This project started in 2013 (Ermacora et al., 2013), in which just a specific emergency-management service was presented. Central core of the service is the special module called Open Data MIssion Planner (ODOMI). This module, exploiting different source of information from open data providers and information thought the Internet (e.g. position and height of obstacles, quality of the internet connection etc.), is able to plan a flight path according to the needs of the required application. When the application is set this module can create a path in order to:

- Avoid connectivity losses and that provides enough signal coverage for the video streaming.
- Finding the best path that avoids obstacles in the environment with enough connectivity coverage
- Finding the safest path e.g. fly only above building, rivers and trees so in case of failure the probability of hurting a citizen in the surrounding environment is relatively low

Robotic Operating System (ROS) (Quigley et al., 2009) is at the base of the cloud platform. ROS is an open-source, meta-operating system for robot software development that is nowadays becoming the de facto standard for robotic software development.

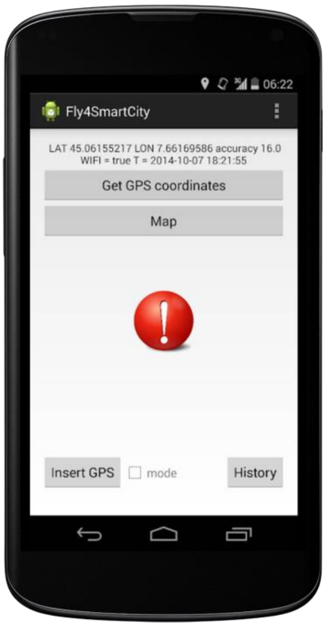
The platform is also designed to be resilient: in case of emergency there is a basic flight stored in the internal memory of the UAV plan with the basic instructions e.g. “land on a certain predefined GPS position” or “return to the home GPS position”. In addition the cloud platform is capable of reconfiguring at any time the plan of any single managed UAV.

3.1. USER INTERFACE

Two ways for interacting with the monitoring and emergency service has been implemented. The first is *via* a smartphone application imagined for the citizen. The citizen in case of emergency can use this application to call the service. The second is a Graphical User Interface (GUI) imagined for managing the drones into the smart city environment from authorized operators (e.g. police officer).

In our experiments the mobile application is an Android application with a simple button for recalling the service. The application calls the service exploiting RESTFUL API over the HTTP protocol. In particular a GET request is sent to the server in the cloud platform.

In the request are present the GPS coordinates of the position of emergency and an unique identification number (ID) for having a history of the requests and improving the managements of the requests. In the figure 20 the mobile application is depicted.



LAT 45.06155217 LON 7.66169586 accuracy 16.0
WIFI = true T = 2014-10-07 18:21:55

FIGURE 17 SMARTPHONE APPLICATION

The second interface, the GUI, was written in HTML5 supported by ROSlibjs, an efficient library merging the low level functionalities of ROS with the high level functionalities and powerful tools of JavaScript.

With this management GUI the operator e.g. the police officer in our example application can manage all the drones in the environment:

- monitor the telemetry of any UAV
- monitor the position of any UAV displayed in the map
- accessing at the video streaming of any UAV
- create a mission according to the requested requirements
- offer assistance thought the interface to the citizen

Figure 21 shows the management GUI.

On the left there is a panel showing the video streaming coming from the interested UAV (on top) and flight information e.g. speed, altitude, battery level.

The right panel gives information about the on-going mission. It shows the starting point of the UAV, the goal position and the path chosen according to the selected requirements (e.g. avoiding obstacles in the environment at the altitude of 10 meters) overlapping the city map.

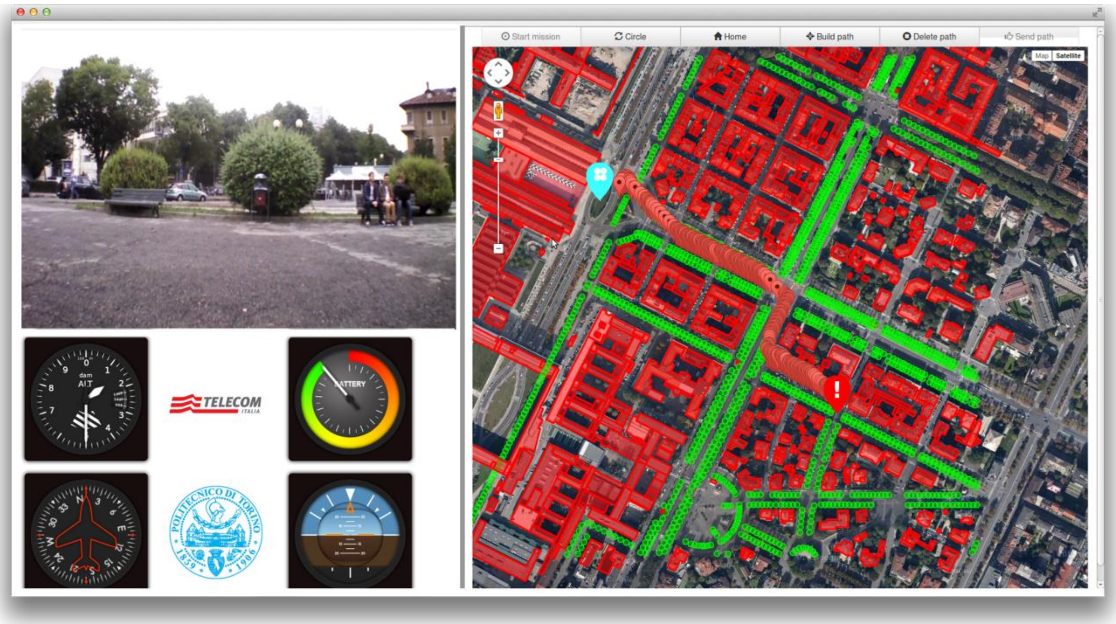


FIGURE 18 GUI FOR DRONES MANAGEMENT

In the top left there are buttons for the management of the mission. In particular is possible:

- to enable the mission. When a request is received the operator can decide to take the mission enabling the drone.
- Circle. When the operator wants to have a more detailed understanding on the surrounding environment of the goal position he can decide to make a circle around that position.
- Home. The operator can decide at any time to terminate the mission and force the UAV to go back to its initial GPS position define as home.

- Create path. The operator wants to monitor a certain zone in the urban environment and creates a set of waypoints (this is a different application of the emergency service).
- Delete path. The operator can delete the path during the phase of choosing the set of waypoints
- Send path. The operator now chooses to send the set of waypoints to the platform. From this moment on the mission starts and the drone will start to fly.

3.2. The Cloud Robotics Platform supporting the service Fly For Smart City

The concepts outlined in previous chapters have practically been implemented as follows:

- *Normal Instance* (NI): a real or virtual machine with high performance.
- *Simple Instance* (SI): a real or virtual machine with low performance.
- *Service Container* (SC): a ROS container (Waibel) identified by its ROS master. The multi-master robot concert technology (http://www.robotconcert.org/wiki/Main_Page) enables ROS container multiplicity.
- *Node* (N): a ROS node.
- *Internal Endpoint* (IE): ROS topic, ROS service or ROS action.
- *External Endpoint* (EE): ROS topic, ROS service or ROS action of a node in another Service Container.

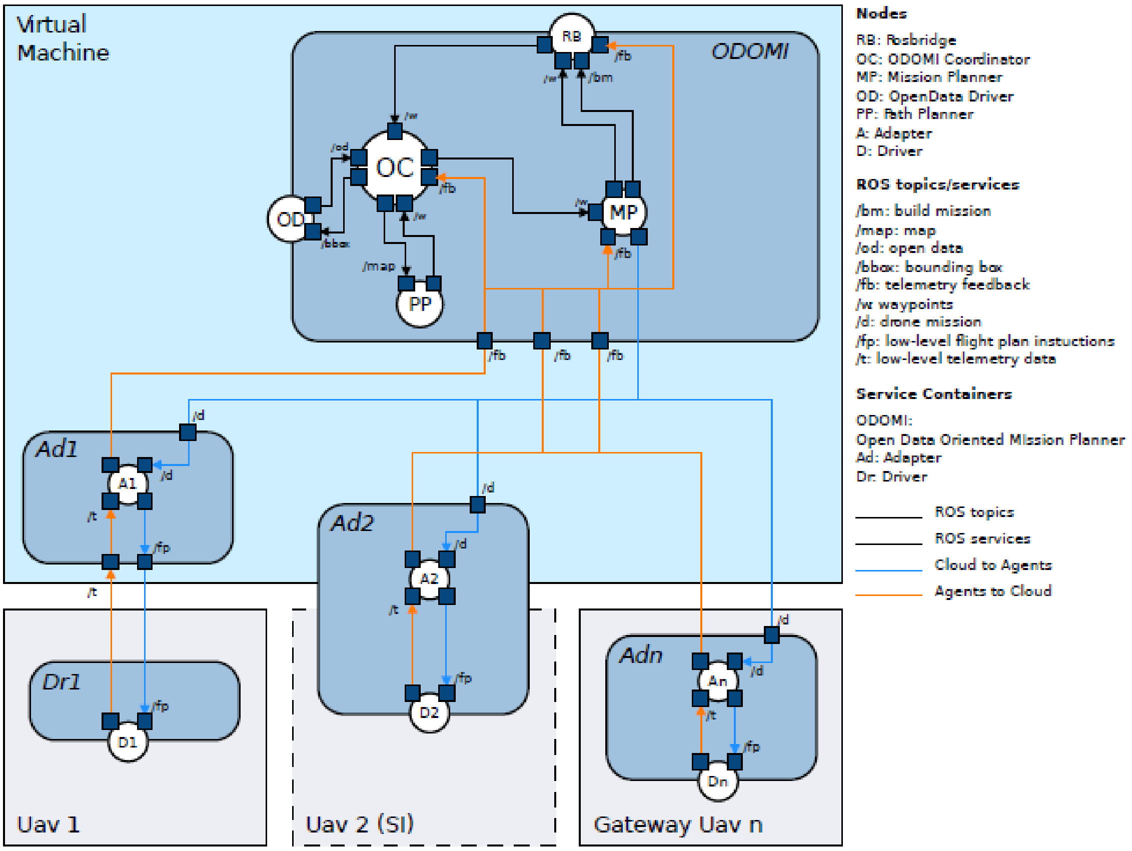


FIGURE 19 THE LOGICAL ARCHITECTURE OF API MANAGERS.

Figure 19 depicts our implementation of the cloud robotics platform. In the picture sharp-edged rectangles stand for NIs, rounded rectangles stand for SCs, dotted-edged rectangles stand for SIs. Any small circle represents a ROS node and its endpoints are symbolized as small squares.

The figure also highlights the difference occurring between *Normal* and *Simple Instances*. In the case of Drone 1, the *Normal Instance* contains a *Service Container* running the driver *node* while the Drone 2 constitutes a *Simple Instance*, since its service container is shared with the normal instance named "*Virtual Machine".*

The Drone n is in this case the *Service Container* (named Adn) runs in a Gateway (ground station). Here both adapter and driver *node* are placed on the same physical machine. The fourth *Normal Instance* presented in the figure, named "Virtual Machine", contains both the ODOMI (*Open Data Oriented MIssion planner*) and the Adapter (Ad1 and Ad2) *Service Containers*. Adapter SCs translate ROS messages coming from UAVs in standard platform messages (i.e. from */sensor_packet_i* endpoints to */sensor_packet* endpoint) and viceversa (i.e. from */drone* EE to */flight_plan_i* endpoints).

ODOMI is the core *Service Container* of the whole platform, and it runs four *nodes*: the rosbridge *node* (the Service API manager described in previous section), the Mission Planner, the Path Planner and the Open Data Driver (described in the next section).

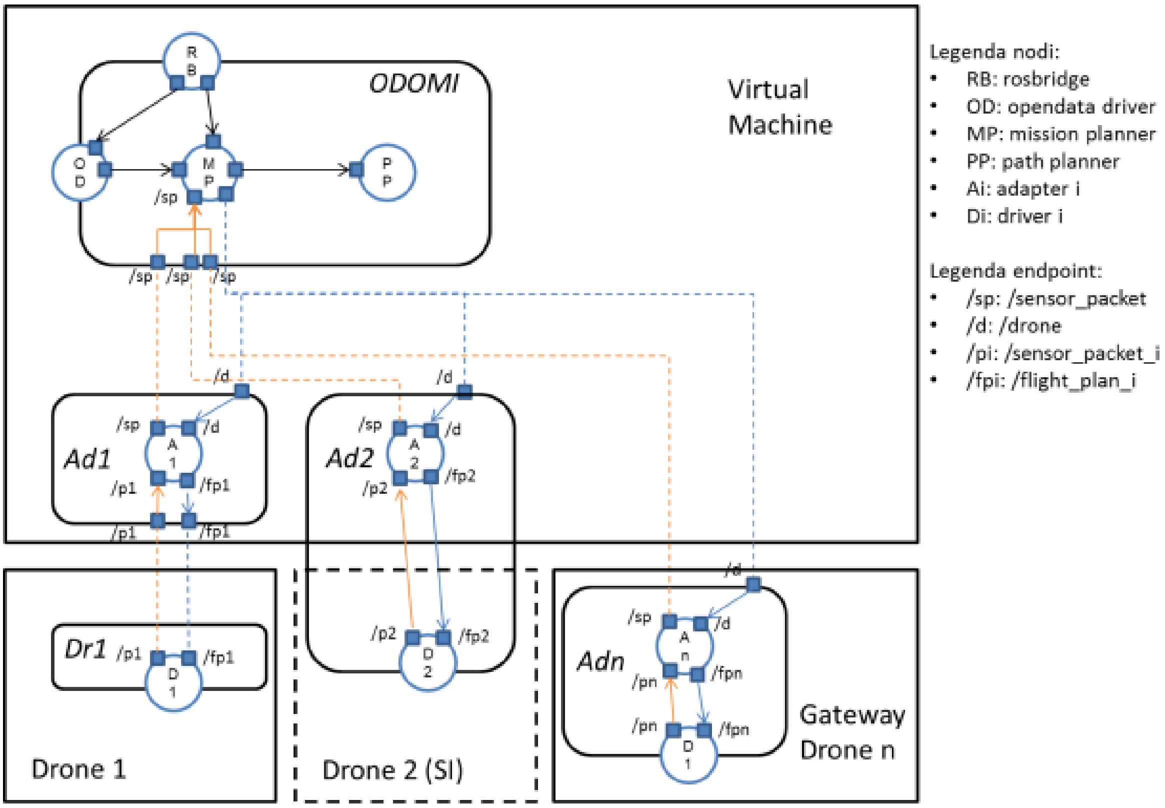


FIGURE 20 THE ODOMI IMPLEMENTATION.

Figure 20, 21 and 23 offers a more functional overview of the communication between each module installed on the platform for the single-UAV case.

Ensuring a secure access to the platform is fundamental. API access is strictly regulated by means of specific security keys provided to the developer after a mandatory registration phase. Moreover instances are connected to the platform via VPN (Virtual Private Network) in order not to be easily accessed by hackers. Another main requirement the platform has to guarantee is concurrency. This is done exploiting the queue mechanisms provided by ROS for multiple-access management. ROS node can either consume a message queue (subscriber), according to computational complexity of consumer algorithm, or produce messages to be queued before sending (publisher), according to available bandwidth.

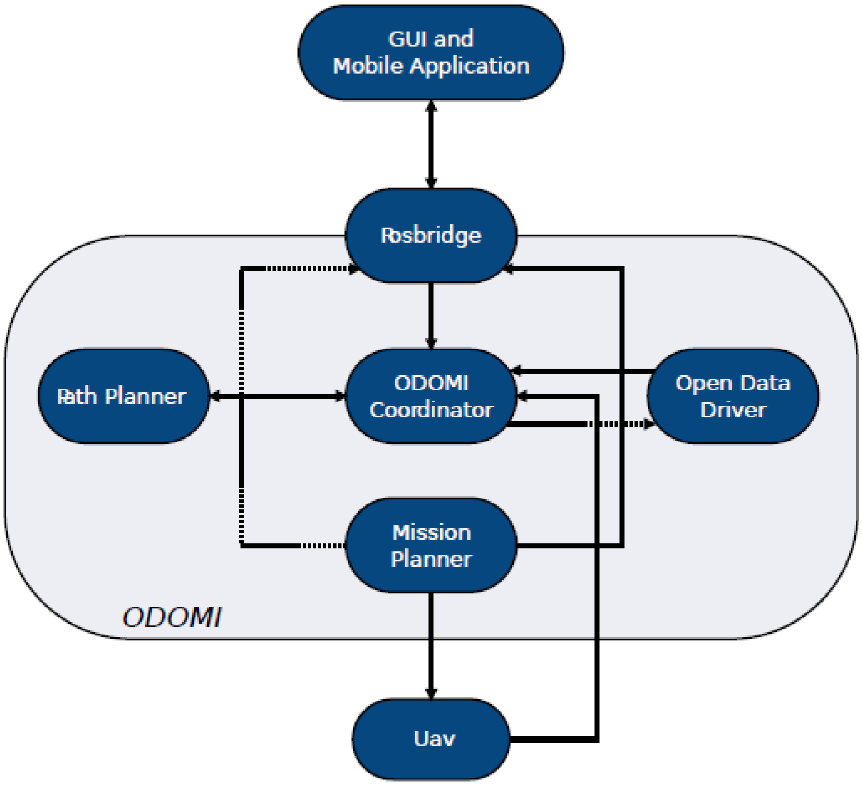


FIGURE 21 ODOMI ARCHITECTURE

3.3. OPEN DATA MISSION PLANNER

ODOMI is the *Service Container* in charge of managing the mission and calculating the path for the UAV. It is the only *Service Container* who has the rights to START, STOP or ABORT the mission.

It is organized in four *nodes*:

- Open Data Driver (ODD): this module retrieves available information by calling the appropriate source of information. It packs the retrieved data into a ROS message and sends it to the coordinator module.

- Path Planner (PP): it receives the cost map created by the coordinator and plans the path (set of waypoints) for the UAV. Further details are in the dedicated paragraph.

- RosBridge (RB): it provides JSON API to ROS interface (Crick et al., 2011). This *node* serves as the interface between the smartphone application and the robotics platform, namely the *Service API Manager* that exposes the APIs for sending the target coordinates that the drone has to reach to the Mission Planner.

- ODOMI Coordinator (OC): this module coordinates all messages in the service container ODOMI. It is coordinates the whole architecture and it creates the mission for the UAV according to the suitable policy.

- Mission Planner (MP): This module takes as input the position of the UAV (home) and the coordinates of the target. It builds then a bounding box message and collects every available information about that area by querying the ODD module. Once it gets these data, it builds a map.

Provider	Data	Response	License
OpenStreetMaps	2D Map		Open Data Commons Open Database License (ODbL).
Open Weather Map	Temperature Wind Humidity	XML/JSON	Creative Commons (cc-by-sa)
Geoportale Torino	2D Map Height Pedestrian areas Rivers and waterways Treed areas Green areas	GML(XML) GML(XML) GML(XML) GML(XML) GML(XML) GML(XML)	Creative Commons public licence" Attribuzione - 2.5 (ITALIA).
5T Torino	Traffic	XML	Creative Commons - CC0 1.0 Universal
	Tram and train lines	CSV	Creative Commons - CC0 1.0 Universal

TABLE 1 OPEN DATA PROVIDERS

3.4. Open Data Driver module

The module that is in charge of retrieving all the information available on the Internet from different sources is the Open Data Driver module (see previous table 1). Geo referenced data (Table 2) are used to give more information to the Mission Planner. Part of these data are actually private and they could not being considered "open" in a strict sense (Foundation). But usually under some constraints they are public (e.g. maximum number of daily/monthly API calls). Therefore this modules taking the current position of the drone and the goal it creates a bounding box. The bounding box is created with those two points plus an arbitrary padding to enlarge the diagonal of the bounding box. This is important for having enough data to download for creating a useful map where the drone can navigate in and it is also important to avoid the download of too many information that could overload the computation phase. Data are shown in Table 1 and 2. It is important to note that custom drivers has to be created for accessing those different data source, since they don't usually have a standard data format. In our case in fact we developed drivers in particular for Geoportale Torino, OpenStreetMaps, 5T Torino, Google Maps and OpenSignal. Once that all the data are successfully retrieved they are sent to the ODOMI Coordinator (OC) that is in charge of creating the cost maps to send consequently to the Path Planner module.

Provider	Data Type	Response	License
Google Maps	Digital Elevation Map	XML/JSON	Google Maps API licensing*
OpenSignal	Average RSSI	XML/JSON	OpenSignal API licensing***
	Tower Info	XML/JSON	

* https://developers.google.com/maps/terms

** http://developer.opensignal.com/terms/

Table 2. Geo-referenced data sources.

3.5. ODOMI Coordinator

This module is in charge of sending the request, with the current position of the drone and the goal, to the ODD module. The ODD module then answers with raw information taken from open data sources. Depending on the chosen policy the ODOMI coordinator module creates a cost map using the data gathered from those open data sources. The cost of any cell of the cost map depends on the policy chosen. Policies are many and can be very different. In general, policies that maximize flight over buildings and trees are thought for real applications in urban environments, in order to minimize the risk of hurting humans or things in case of failures. Then the Path Planner module will use this cost map for creating the optimal path.

3.6. Path Planning module

Path planning or *trajectory planning* is a well-known problem in mobile robotics, and it has been recently applied to UAVs. Path planning is one crucial task for UAVs that have semi-autonomous or autonomous flight capabilities. As technology and legislation move forward, the need arises for an increase in their level of autonomy. Therefore, the focus is moving from system modeling and low-level control to higher-level mission planning, supervision and collision avoidance, going from vehicle constraints to mission constraints (Gutierrez, 2006.).

UAVs present some peculiar characteristics, like their flight dynamics, 6DOF movement, and high levels of uncertainty in their own state knowledge, as well as limited sensing capabilities (Goerzen, 2009). However, other problems have already been addressed by the robotics community, like partial knowledge of the environment. The autopilot is in charge of low-level vehicle control. Some work has been carried out in (Jun, 2003) for path planning in presence of uncertainties and with signal constraints in (Grotli, 2012), (Grancharova, 2012), (Chi, 2012). Most of path planning approaches rely on a two-stage procedure:

they first solve what is called the *global path planning* problem, by finding a feasible trajectory given the current pose of the agent, the destination and a map of the environment. Then they implement a control strategy called *local path planning* to follow the found trajectory (Goerzen, 2009). In our work we focus on global path planning, since our objective is the generation of a trajectory, which will be decomposed into a series of intermediate waypoints, which in turn will be fed to the UAV's autopilot.

In our application, we also assume that the UAV can only fly at a fixed altitude. The trivial flight strategy for an UAV in order to reach a certain goal would be to fly to a certain altitude above buildings and follow a straight line. Since in our application we exploit crowd-sourced information about LTE connectivity and this information is mostly gathered by users on the streets, the UAV has to fly at a lower altitude, in order for the connectivity data to be meaningful.

Based on the assumption of having a fixed altitude, we simplify the problem to the 2D case and we use the implementation of the *D*-Lite* algorithm described in (S Koenig). D*-Lite is a fast implementation of the D* algorithm, and is based on *Lifelong Planning A** (LPA*) (S. Koenig and M. Likhachev T. Dietterich).

D* is an incremental heuristic graph search algorithm which is able to deal with incomplete information and observations. D* algorithm ensures completeness and optimality properties.

The Path Planner (PP) module is implemented as a two-level planning approach. Given a bounding box area of operation, we first build a *static cost-map* where the traversal cost for each cell depends on fixed or low-changing open data, like the presence of tall buildings or structures. Any structure with an average height which is higher than the flying altitude makes the cells belonging to that area untraversable. This information is given by the ODD module.

Given the current position of the UAV and the desired goal, we calculate a feasible path using D*. We then extract a series of waypoints from the path using a simple procedure. First we calculate the distance transform of the map. For each cell p we calculate the distance of the nearest obstacle as

$$D_P(p) = \min_{q \in P}\big(d(p,q)\big),$$

where P is a grid of points representing the map and $d(\cdot)$ is the distance between two points. Then, for each waypoint we expand a circle around it with a radius of value $D_P(p)$, where p is the cell corresponding to the waypoint. The next point of the trajectory that lies on the circumference is taken as the next waypoint. The process is then repeated until the last point of the trajectory. With this procedure we extract a list of waypoints that are given to the UAV's autopilot.

While the UAV is flying, we also create a *dynamic cost-map*. The dynamic cost-map is built based on fast-changing or dynamic open data. The rate at which this dynamic cost-map is updated varies depending on the estimated changing rate of the particular open data. Each time new open data is available, a new dynamic cost-map is created. In our application we use LTE *received signal strength* (RSSI) heat-maps. This information is again given by the ODD Module. The cost of each cell is based on the RSSI for that particular cell; if the RSSI is below a critical threshold, the cell is marked as obstacle, since in our application we cannot afford to lose connectivity. Each time a local cost-map is available, the path is calculated again starting from the next waypoint to be reached until the goal with the addition of the dynamic cost-map data.

In Figure 22 it is showed the generation of the static path in a simulated case. The minimum path is found from the starting point to the goal while avoiding obstacles. In Figure 23 it is showed how the trajectory changes with the addition of a dynamic cost-map.

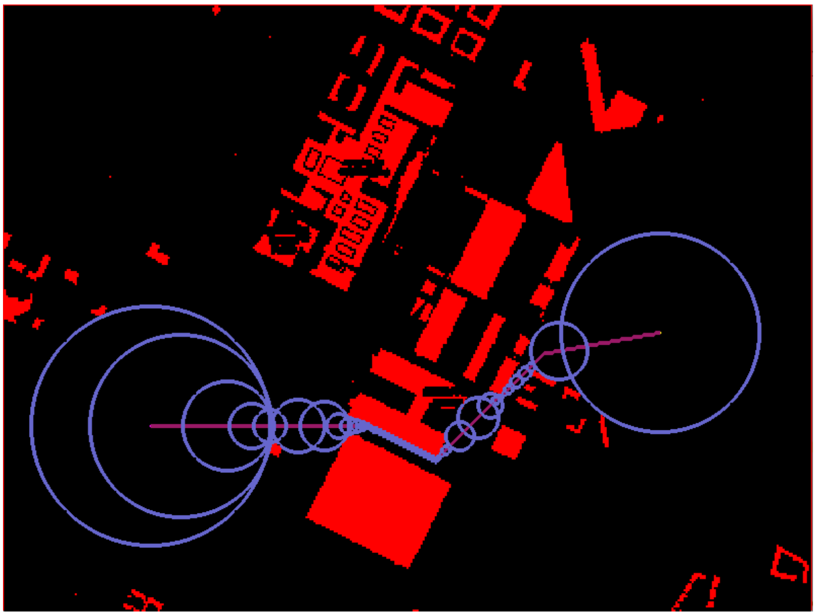


FIGURE 22 GLOBAL COST-MAP. THE STARTING POSITION IS SHOWN IN YELLOW; THE GOAL IS SHOWN IN BLUE; OBSTACLES ARE SHOWN IN RED; THE FOUND TRAJECTORY IS SHOWN AS A PURPLE LINE AND THE WAYPOINTS WITH ASSOCIATED RADIUSES ARE SHOWN AS CIRCLES.

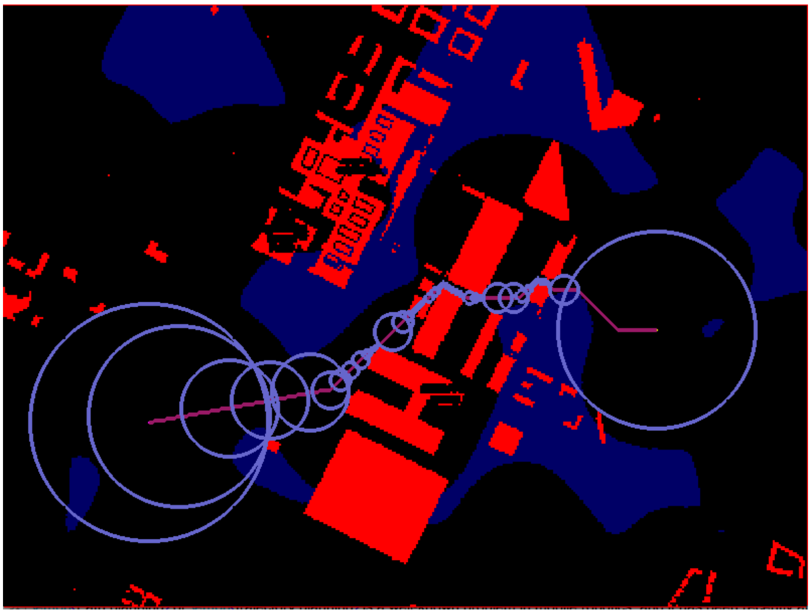


FIGURE 23 GLOBAL AND LOCAL COST-MAP. THE LTE AREAS WITH LOW RSSI ARE SHOWN IN BLUE

We also show how the Path Planner can force the UAV to fly on areas which are considered “safer” from the citizen’s point of view, like water surfaces. In Figure 24 the UAV has to fly over a park, which is crossed by a river. Since we assume that areas over water surfaces are safer and free from obstacles, we assign a lower cost to the relative cells. In our example, the UAV favors flying over the river, when possible.

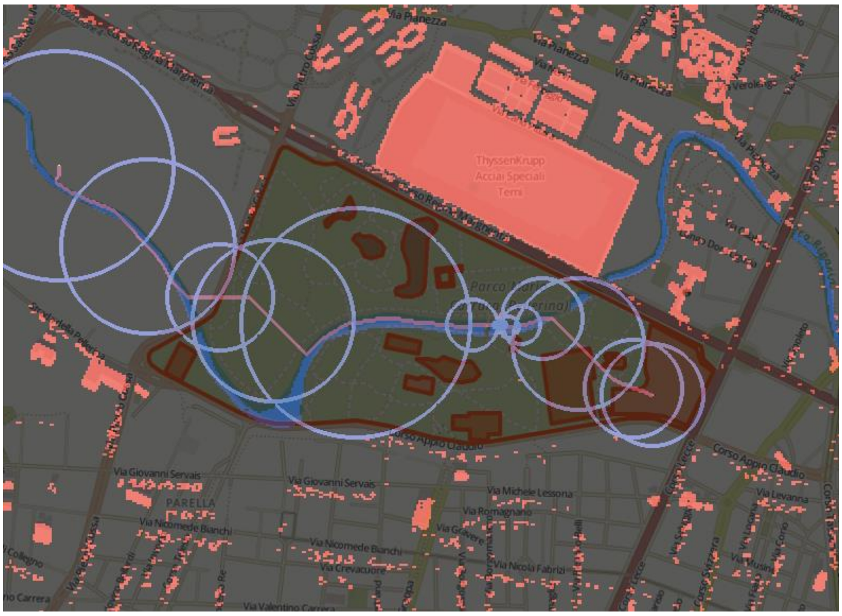


FIGURE 24 GLOBAL AND LOCAL COST-MAP. THE LTE AREAS WITH LOW RSSI ARE SHOWN IN BLUE.

3.7. The agents

A quadrotor has been chosen as the sole agent for validating the overall system in its first stage. It offers a series of advantages with respect to other possible architectures (e.g. unmanned ground vehicles, fixed wing autonomous aircrafts etc). Its VTOL (Vertical Take Off and Landing) capabilities allow for quick take-off and landing operations in busy or space-constrained places. The possibility to hover above the target makes it well-suited for surveillance and monitoring tasks. Moreover it offers a simpler mechanics and requires less maintenance with respect to a standard helicopter offering similar autonomy and payload. However a quadrotor is an inherently unstable system (Hoffmann, 2007), and for this reason it requires fast on-board controller to guarantee its stability in standard flights (Bouabdallah, 2005). This controller typically runs on a dedicated electronic board, equipped with a number of inertial sensor (accelerometers and gyroscopes) to detect the attitude angles of the quadrotor frame and its rotational velocities; barometers and sonar sensors (for altitude measurements), GPS receivers (for autonomous waypoint navigation) and magnetometers (providing an estimate of the heading of the aircraft) are other common devices, often mounted on the same board.

In this project the validation of the proposed cloud robotics service has been conducted using a Parrot AR Drone and Mikrokopter (http://mikrokopter.de/en/home - system, developed by HiSystems GmbH in Germany.

Born as an open-source project Mikrokopter evolved in a complete amateur-class platform for control and operation of multirotors. The term *Mikrokopter* refers both to the ready-to-fly multirotor solution and to a set of electronic boards that can purchased individually and mounted on a custom frame; hereafter the word Mikrokopter (or MK) will only indicate the latter meaning.

The core of the system is composed by two boards. The *FlightControl* board guarantees vehicle inherent stabilization and altitude-hold function. It relies on Atmel ATMEGA644 board running at 20MHz and interfaces with the main inertial sensor. The *NaviControl* board adds a set of GPS/Compass based autonomous navigation functions (waypoint navigation, come-home function, position hold mode).

MK allows the user to take external control of the UAV (i.e., bypassing the radio controller) by means of a dedicated serial protocol but, in spite of its open-source origins, interfacing with MK is not a trivial task, mainly because of a substantial lack of documentation and because of the barely readable source code for non-German speaking people (Sa, 2012); presents a complete reverse-engineered description and model of the system.

In consequence of the poor support offered to the developers, ROS support for Mikrokopter appears to be limited to an outdated ROS stack (https://wiki.qut.edu.au/display/cyphy/MikroKopter+ROS+nodes) that is no longer maintained. Therefore a ROS interface implementing the fundamental input/output methods useful to communicate with the autopilot on its serial protocol have been written from scratch. An overview of the messages is reported in Table 3.

Type	**Topic name**	**Message Type**	**Message Struct**
pub/sub	/parameter	`open_data_msgs/Parameter`	*string* key *string* value
pub/sub	/polygon	`open_data_msgs/Polygon`	*Coordinate[]* vertex
pub/sub	/data	`open_data_msgs/Open_data`	*int* type (static, dynamic) *string* label (nome open data) *Parameter[]* attributes *Polygon* area
service	**/bounding_box**	`open_data_msgs/BoundingBox`	*Polygon* bounding_box *string* label (vuoto = tutti) *int* type -- *bool* resp
pub/sub	**/open_data**	`open_data_msgs/Data`	Open_data[] data
pub/sub	**/mission_map** **/dymanic_map**	`mission_planner_msgs/MissionMap`	*Header* header *Coordinate* home (gps) *Coordinate* target (gps) *Coordinate* origin (gps) *Float* resolution (m/pix) *int8[]* data

pub/sub	**/drone**	`mission_planner_msgs/Drone`	*Header* header *string* name *string* type_name *Coordinate* home *Move[]* movements
pub/sub	**/sensor_packet**	`mission_planner_msgs/SensorPacket`	*Header* header *Coordinate* current_position *int16* power_data …
service	**/target**	`path_planner_msgs/Position`	*Coordinate* position --- *bool* resp
pub/sub	**/waypoints**	`mission_planner_msgs/Waypoints`	*Coordinate[]* waypoints

TABLE 3 A COMPLETE OVERVIEW OF THE ROS-MESSAGES EXCHANGED AT PLATFORM LEVEL.

3.8. Experiments

We made our experiments in different simulated and real scenarios. In this chapter we report some of the most interesting results. First two tests are in a simulated scenario where show the integration of different source of open data in ODOMI. The third is a real experiment with ODOMI, the PP module and the A.R. drone commercial quadcopter. The last test shows an event, organized in collaboration with Telecom Italia that was demonstrating the system composed by GUI and cloud platform using a commercial quadcopter in a private area (aereoport).

In the simulated tests we took real data from open data from open data (buildings perimeter and height, trees and waterways) and crowd-sourcing using a mobile Android app (3rd Generation (3G) and 4th Generation (4G) coverage). In the simulated experiment all the stack from the open data driver to sending the path to the drone is real. The only difference was that we were not flying for real the drone but only generating the path. It has to be noticed that was not easy to perform a real experiment because of the current legal restrictions and security reasons. The system is implemented in C++ and Python on GNU/Linux. Tests have been carried out on a common laptop. Source code is available at (https://github.com/fly4smartcity/odomi) and the GUI (https://github.com/fly4smartcity/rendezvous).

3.9. Experiment 1

In this first experiment we simulated a simple planning strategy for flying an UAV in an urban area. The Path Planner is in charge of minimizing the path length, while avoiding obstacles. The simulated A.R. Drone can flying at the maximum altitude of 6 meters with its GPS receiver. Therefore in this simulation building and trees higher of this altitude are considered as an obstacle. Results are shown in Figure 25. In this picture are shown also the initial position of the drone, the final and the extracted waypoints with their markers.

3.10. Experiment 2

In this second experiment there is the comparison between three different planning strategies. Those three strategy, as in the previous case, try to minimize the path length. In the first test the strategy is to avoid obstacles, in particular buildings and trees higher than the maximum altitude at the drone can fly. In the second test the strategy is affected to the risk factor: the system tries to find a tradeoff between minimizing the path length and risky areas. The third test has the same strategy but in addition the system tries also to minimize zones where the 4G RSSI is low considering those zones as well as “risky”. Figure 26 shows the result.

3.11. Experiment 3

In this experiment the system is running in a real use case with a low-cost platform.

The drone is an A.R. Drone Parrot 2.0 equipped by Flight Recorder, the GPS module. The GUI is accessible by a laptop connected thought a 4G network to the cloud robotics platform. The laptop and the Parrot communicate via Wi-Fi connection. On the GUI it is displayed the initial position of the drone. So the user select the goal position and consequently the waypoints are sent to the guidance system. In Figure 27 are shown the results. Since the GPS module on the Parrot has an error of 5-10 meters we inflated the obstacles by 5 meters. It is interesting to notice that since the GPS is not enough accurate and the harsh wind dynamic against the drone, the Parrot is not able to follow exactly the path but still it is able to avoid obstacles. In Figure 28 is shown the experiment in progress . As a result we can see that the Parrot reached the goal with a large tolerance of 10 meters due to the bad accuracy of the GPS and localization in narrow areas.

3.12. Experiment 4

Here a real live demonstration in which the entire system is tested. In this case a commercial drone was used, with a MicroPilot's MP2128g autopilot for the in-flight stabilization of the drone. The experiment was carried out in Aereo Club Torino's airport in Turin, Italy. The UAV was communicating both to a ground station via Radio Frequency 5.8 Ghz radio link and to the cloud robotics platform via 4G LTE connectivity. For security reasons and as required by the Italians legislations a pilot was supervising the flight of the drone. So in this experiment the drone was always flying at line-of-sight. In addition the flight zone was closed and public was kept at a distance of 150 meters. The live experiment was divided in two test-cae.

The first one was the emergency service. A person was playing the part of the citizen in emergency calling the service with the smartphone application. Then an authorized user authorized the flight thought the GUI. When the drone reached the desired position the user performed a circle around the goal position and looking at the real time video streaming he could have a better understanding of the surrounding environment (Figure 29).

In the second test the authorized user selected a set of waypoints on the GUI, clicking on the map. Then the user started the mission and the drone started to fly and sending the real time video streaming and telemetry to the GUI. In Figure 30 is shown the second experiment. Both experiments has been authorized by ENAC (Ente Nazionale Aviazione Civile), the Italian civil flight authority. Videos of those live tests are avavilable at (http://tinyurl.com/l5lbry4, http://tinyurl.com/qd2whr5).

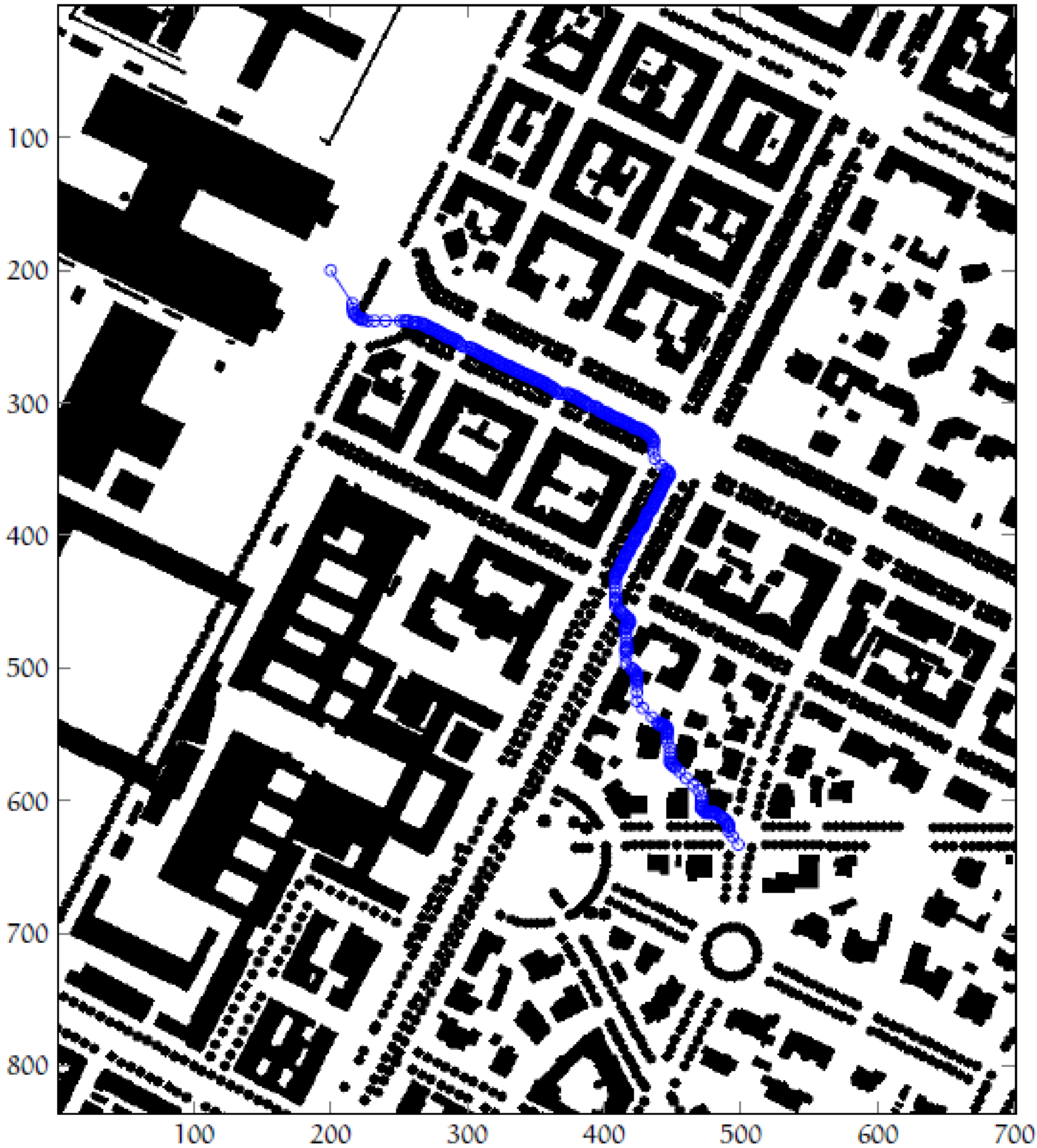


FIGURE 25 EXPERIMENT 1, SIMULATED EXPERIMENT. OBSTACLES FROM OPEN DATA ARE SHOWN AS BLACK CELLS; COMPUTED WAYPOINTS ARE SHOWN IN BLUE. THE STARTING POINT IS IN THE TOP-LEFT AND THE GOAL IN THE BOTTOM-RIGHT.

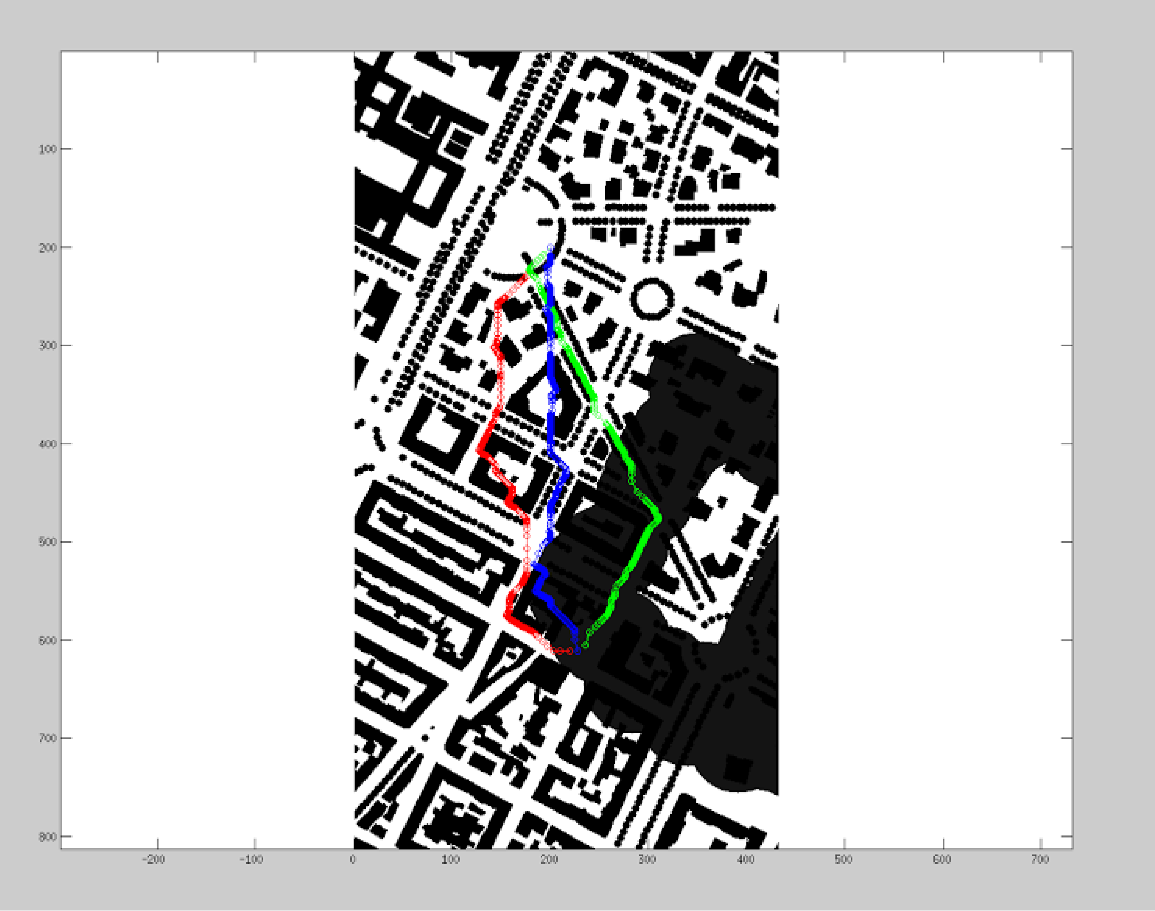


FIGURE 26 COMPARISON BETWEEN THREE PLANNING STRATEGIES. BLACK CELLS REPRESENT OBSTACLES, GREY CELLS REPRESENT AREAS WITH NO 4G CONNECTIVITY. GREEN LINE: OPTIMIZE FOR SHORTEST PATH, AVOID OBSTACLES, IGNORE CONNECTIVITY; BLUE LINE: OPTIMIZE FOR SHORTEST PATH AND MAXIMIZE TRAVEL OVER ROOFS AND TREES FOR

SAFETY, IGNORE SIGNAL; RED LINE: SAME AS BLUE LINE, BUT PREFER AREAS WITH 4G CONNECTIVITY. THE STARTING POINT IS IN THE TOP-LEFT AND THE GOAL IN THE BOTTOM-RIGHT.

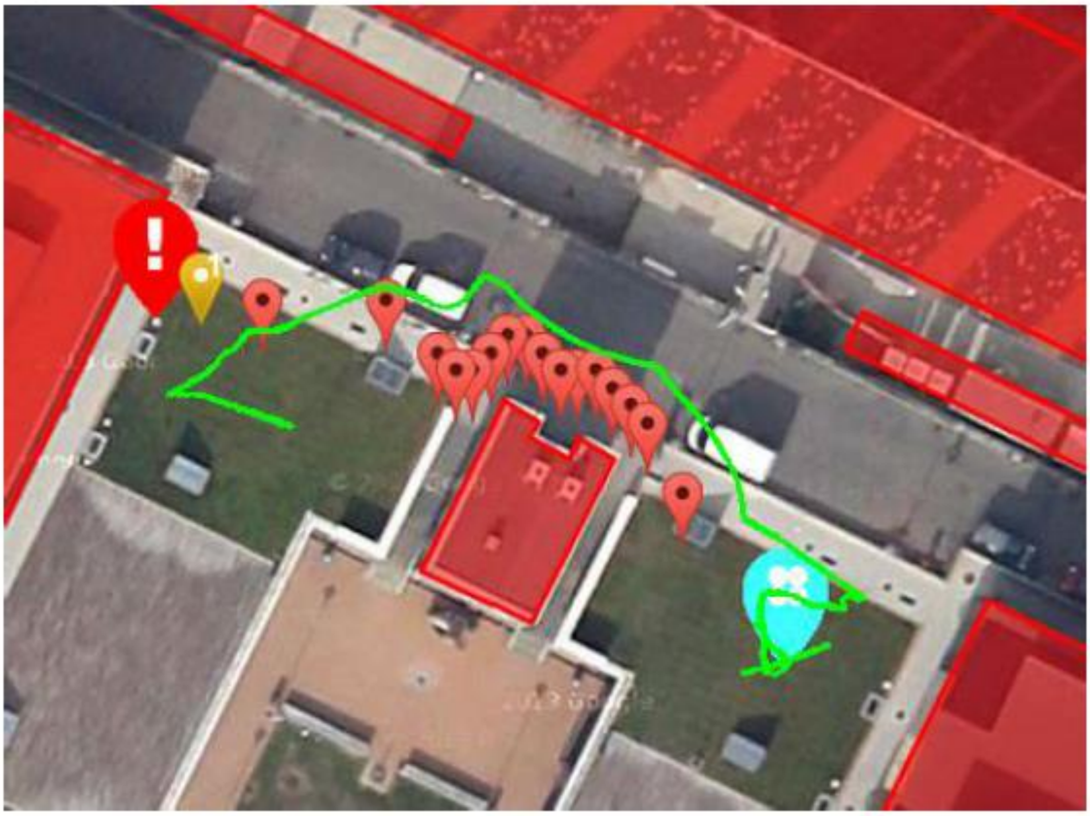


FIGURE 27 REAL TEST OVER A SHORT TRAJECTORY. OBSTACLES FROM OPEN DATA ARE SHOWN IN RED; STARTING POINT AND FINAL GOAL ARE SHOWN AS MARKERS; COMPUTED WAYPOINTS ARE SHOWN AS RED MARKERS. REAL TRAJECTORY FOLLOWED BY THE UAV IS SHOWN IN GREEN.

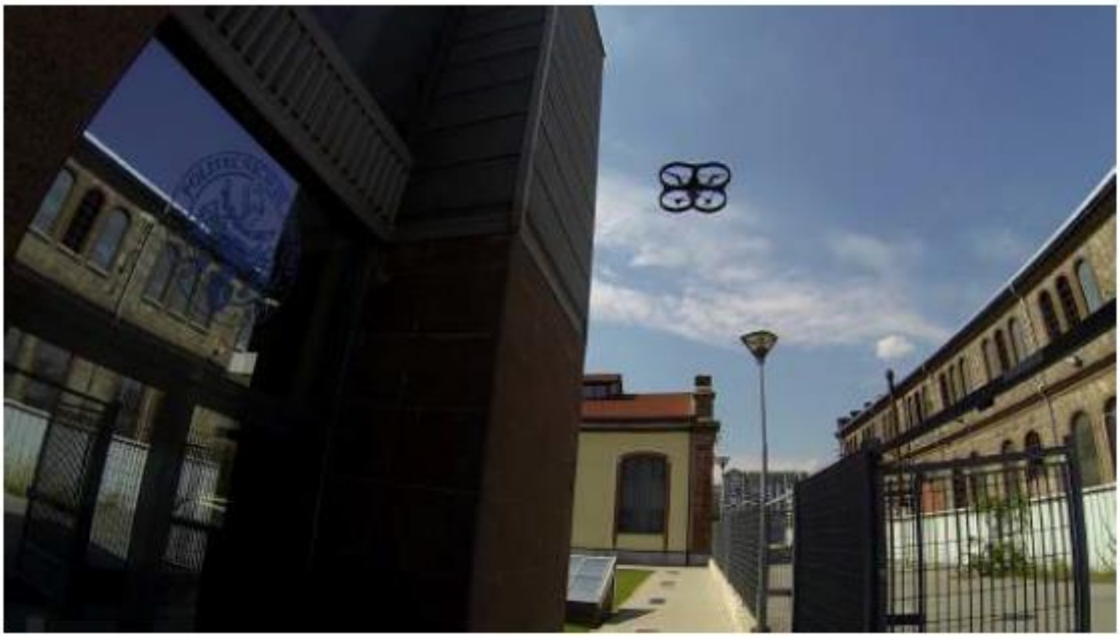


FIGURE 28 REAL TEST OVER A SHORT TRAJECTORY. PHOTO OF THE EXPERIMENT

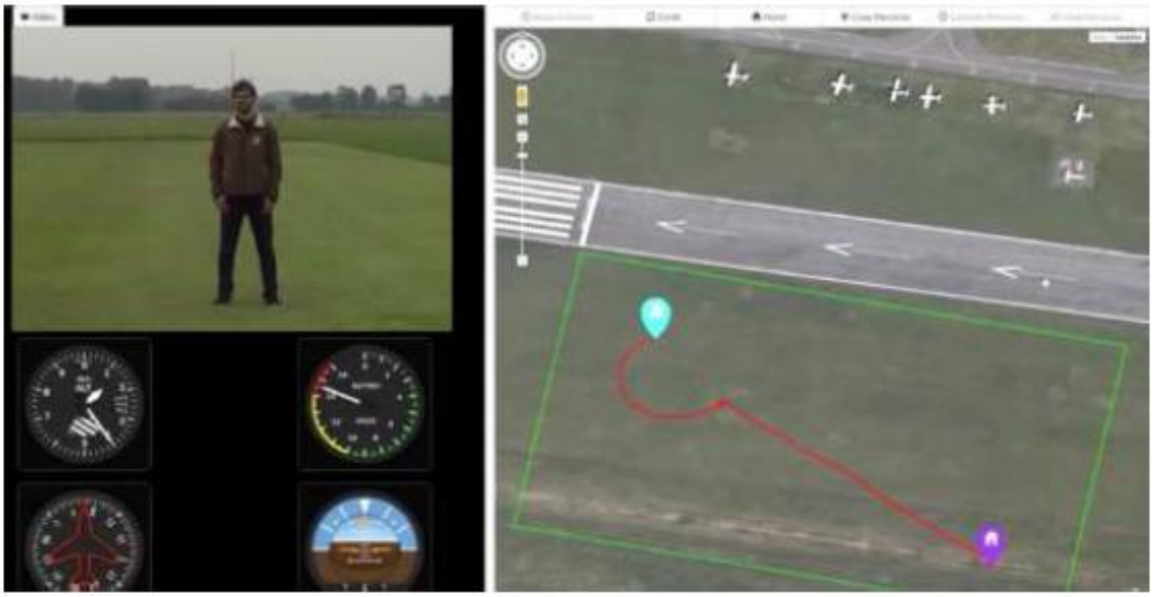


FIGURE 29 VIDEO STREAMING AND TELEMETRY DURING THE LIVE TEST (FIRST TEST-CASE).

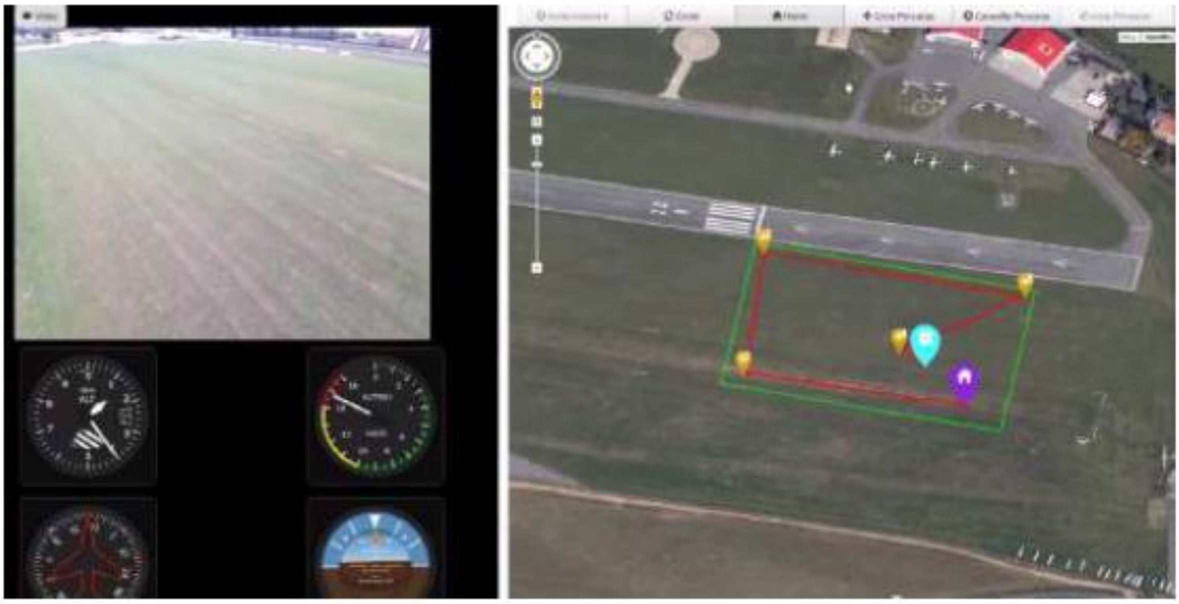


FIGURE 30 VIDEO STREAMING AND TELEMETRY DURING THE LIVE TEST (SECOND TEST-CASE).

3.13. CONCLUSIONS

Fly4SmartCity is a large-scale cloud-robotics service that aims to prove that a real world application of cutting-edge technologies in robot-networking and unmanned vehicles is actually possible. The goal of F4SC is to provide a service for emergency management in smart city environment; this is achieved by the strict integration of:

- A capable, reliable and stable cloud platform;
- Sufficient agile autonomous agents;
- A high-bandwidth, low-latency mobile network;
- A rich set of data, drawn from various providers;
- A well-documented and supported robotics middleware.

A ROS-based cloud-robotics platform has been expressly developed to support the service and the main concepts of its implementation have been discussed. Then the service itself has been presented and the software modules described from a functional point of view, together with the messages they exchange.

Several major technical issues (e.g. autonomy of the UAV) and legal restrictions imposed by the civil flight authorities (Regolamento Mezzi Aerei a Pilotaggio remoto) still prevent a common and practical use of the infrastructure presented, however we have proved here how the data about the urban fabric, that are in large part public in a smart city, offer a valuable source that it is possible to exploit in order to guarantee safer flights and operations. We showed how these information can be integrated into a simple 2D planning module that is able to generate feasible flight trajectories that are also complaint with safety and signal quality constraints (e.g., fly over rivers when possible, avoid crowded areas, avoid areas with low mobile signal strength, etc.).

Finally, the cloud platform has been designed not to be tied with the described service. On the contrary it offers support to third-party developers by providing management and platform APIs and, although it is not publicly available at the moment, the possibility of releasing it as an open-source project is under investigation.

4. Sliding Autonomy in Cloud Robotics Services for Smart City Applications

In the previous chapter there is a deep explanation of Fly4SmartCity. Fly4SmartCity is a project that wants to apply the use of small Unmanned Aerial Vehicles for providing different services in a smart city environment. In fact this is possible with the strict integration of a cloud robotics platform with UAVs supported by 4G LTE connectivity. Internet and connectivity play a key role in the robotic platform for managing UAVs. The cloud robotics platform manages UAVs using "*shared knowledge*" of the smart city environment. Data source of geo-localized information present in the Internet is what we call *Shared knowledge.* In particular there are two main types of data source: the first is an external source e.g. Open Data expressing the position of buildings, trees and rivers or LTE signal strength. The second is an internal, sometimes private, source coming from data gathered by other UAVs that wants to improve and maintain updated the external source. For example when the service is required the cloud platform generates a cost map based on the information coming from the *shared knowledge* e.g. physical obstacle geo-localized, LTE signal strength, experiences of previous path, preferred path. The cloud robotics platform then manages and controls the flight of UAVs according to the Level Of Autonomy (LOA) of the service (central topic of this chapter). In particular, in this chapter there is perspective towards different scenarios of services better explained in the next paragraphs: Emergency Management Service, Monitoring Service, Mixed Reality Teleoperation and Mixed Initiative Service, and finally Crowdsourcing Service.

4.1. Sliding Autonomy

The concept of Sliding Autonomy is popular and proposed in literature (Scerri, July 2002) , (Heger, 2006), (Sellner, July 2006.). The basic idea is to combine the advatntages of different Levels of Autonomy sliding from a level to another or as in this chapter applying different services with different Level Of Autonomy. Here mainly three Levels of Autonomy are considered:

- full autonomy,
- mixed-initiative
- manual teleoperation

This idea was mainly inspired by (Carlone L. et al., 2010) and (Sauer., 2010) where a sliding autonomy framework was described especially for improving situational awareness, decreasing the workload of the human operator and meanwhile guaranteeing safe operations. Therefore we describe in the next paragraphs all the possible cloud robotics services for smart city application exploiting the sliding autonomy approach. The key agent, so the entity, that interacts with users and manages different levels of autonomy and the passage from one to another is called Sliding Autonomy Module (SAM). SAM is conceived as a software module in the cloud platform that supports all the cloud services and has an internal representation of the Levels Of Autonomy as a finite state machine.

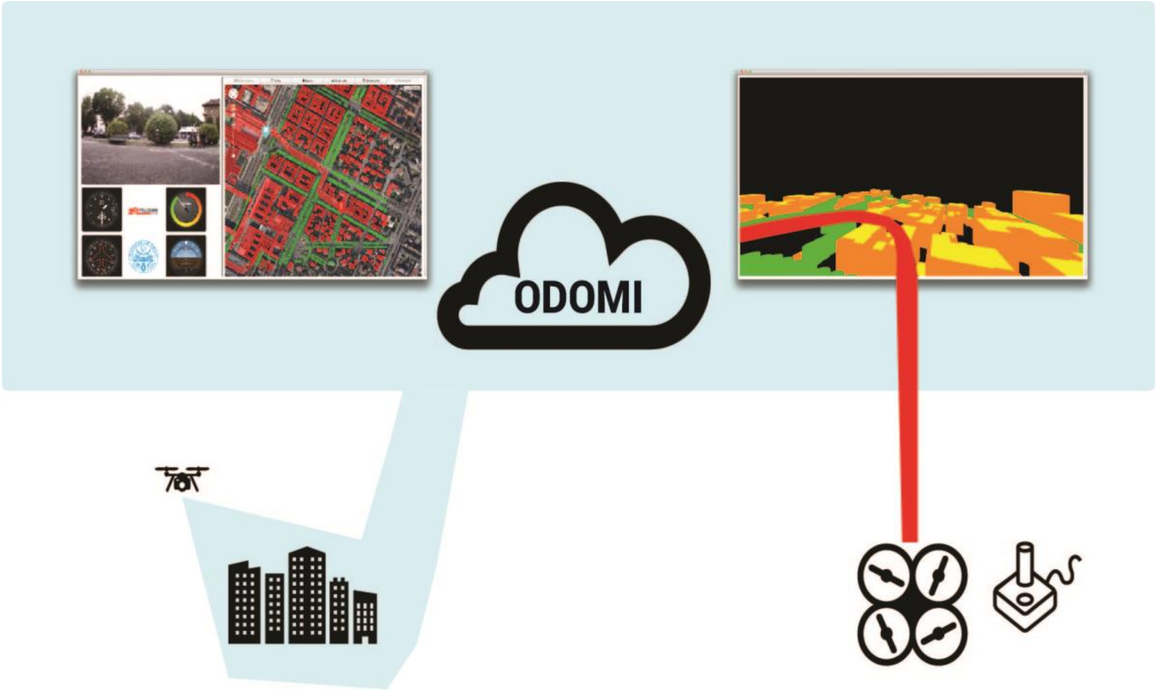
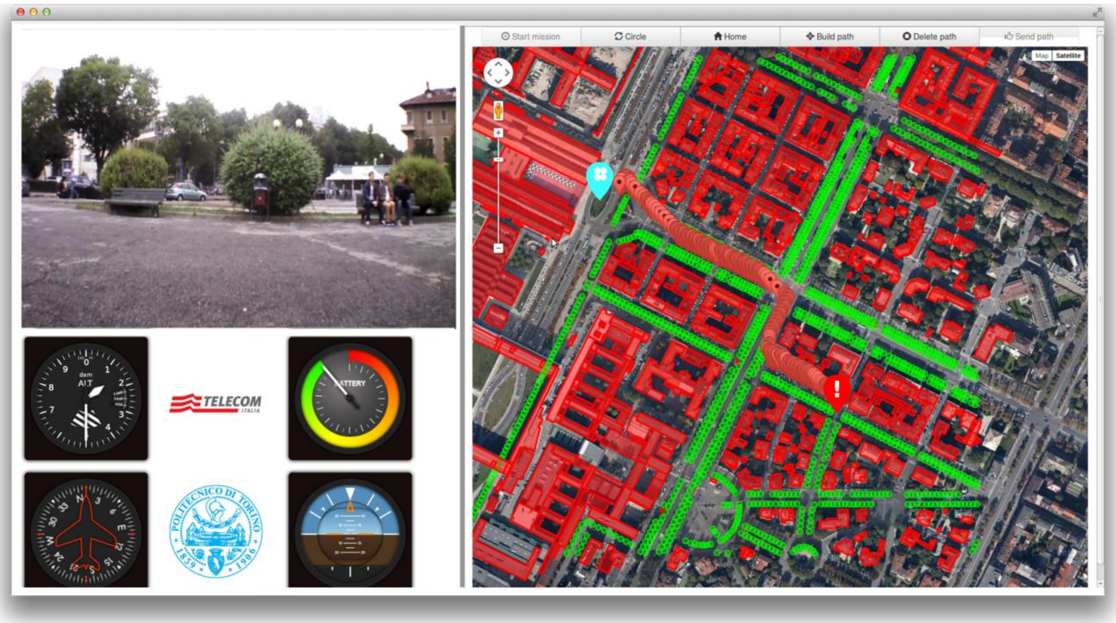


FIGURE 31 IN THIS FIGURE THE MAIN SERVICES DESCRIBED IN THIS DISSERTATION: EMERGENCY MANAGEMENT SERVICE, MONITORING SERVICE (TOP LEFT), MIXED-REALITY TELEOPERATION AND MIXED-INITIATIVE SERVICE (TOP RIGHT/BOTTOM RIGHT) AND CROWDSOURCING SERVICE (BOTTOM LEFT). IN THE MIDDLE THE ODOMI MODULE, EXPLAINED IN THE PREVIOUS CHAPTER, REPRESENTING THE *SHARED KNOWLEDGE* ESSENTIAL FOR SUPPORTING ANY OF THOSE CLOUD SERVICES.

4.1.1. Emergency Management Service

As described in the previous chapter, a citizen through the use of a smartphone application ask for the service of emergency management by sending a request to the cloud platform. Then, on the cloud platform side, the human operator authorizes the request. Then the cloud platform selects the best suitable UAV and calculates the Path Planning (PP). In the end the UAV reaches the position where the emergency occurred. As outlined before the *shared knowledge* source it is used by the Path Planning (PP) module for building a reliable cost map and consequently the UAV path. The way to attribute costs in the cost map depends on the policy required by the user. For example a policy could be that the UAV must to fly at low altitude avoiding obstacles, or the flight path has to prefer at higher altitude over buildings and trees in order to minimize the risk of hurting humans or damaging properties in case of failure. When the drone is in the requested position the human operator can perform a circle around the goal position or switching to a completely manual mode (teleoperation). Once that the operation is finished the operator can decide to return to home position automatically.

4.1.2. Monitoring Service



The monitoring service is mainly imagined for environmental monitoring (e.g. rivers inspections for flooding preventions). In this case the human operator selects some way points on the map and can or not exploit the *shared knowledge* for having more information regarding the surrounding environment. Then the cloud platform chooses the best suitable UAV, creates the flight plan and send it to the chosen UAV. The flight is completely autonomous but the user at any time can switch to teleoperation mode or activate the Mixed reality Teleoperation and Mixed Initiative Service.

4.1.3. Mixed-Reality Teleoperation and Mixed-Initiative Service

As in the previous services, the Mixed-reality and Mixed-initiative Teleoperation Service is supported by shared knowledge present in the cloud robotics platform. So a teleoperation service is possible with visual information regarding obstacles, low LTE signal strength and suggested path to follow, with the current position in the path and future intended positions. All of these visual information are displayed using a mixed-reality approach on the teleoperation GUI. Although this service is meant to support the teleoperation during the monitoring phase the user can switch to a semi-automatic mode called Mixed Initiative mode. This mode becomes important when it is useful to decrease the workload of the user during monitoring operations. A similar approach is explained in (Sauer., 2010). This mode sees the UAV following a pre-determined path but the user can still modify the position of the robot with a restricted teleoperation. As in (Cacace, 2014) the degree of autonomy changes between the human operator and the robot managed by the SAM entity. For increasing the safety there is a low level reactive Obstacle Avoidance module in order to avoid any possible collision. Although the user would send a dangerous command for the UAV or the environment this module tries to apply a strategy for avoid any possible collision.

4.1.4. Crowdsourcing Service

As explained in the introduction the Crowdsourcing Service has to improve and maintain the *shared knowledge* source updated and consistent. The main idea is that Crowdsourcing Service gathers pictures and LTE signal strength from users around the city, drones and over the internet. There is a Task Allocation module that organizes UAV missions in the smart city, choosing the best suitable UAV and planning the mission to the desired goal. For example the platform can decide to send a UAV in a certain particular zone because the information in that zone is not present or not updated. This module as the other modules exploits the Path Planning module. Consequently these new data are geo-localized, matched and attached to the current pre-existing source of *shared knowledge* using also 3D mapping algorithms and reconstruction. On the citizen's side we imagine they having a smartphone application (similar to e.g. OpenSignal https://opensignal.com/android/) and continuously logging the LTE signal strength geo-localized. Then we imagine a huge amount of pictures gathered from users and over the Internet (e.g Flickr, Instagram, Wikipedia) for reconstructing the urban environment. A similar and inspiring approach is described in (Snavely, 2006). In this approach the services manages UAVs automatically while the human operator has only a supervisory control. In case of failure the human operator is notified by the SAM agent. In order to solve the necessity of operating manually the human operator can take the control switching to teleoperation mode or Mixed-Reality Teleoperation and Mixed-Initiative Service. Once the necessity is solved the human operator can return to supervisory mode. In addition the human operator can plan manually a mission in a particular zone if in need, again changing the status from Supervisory Control to another service.

5. The User's Flying Organizer (UFO) project

As explained previously this dissertation has the purpose of analyzing all the possible ways to interact with drones. A good approach is explained in the previous chapter where a Sliding Autonomy paradigm is outlined. Then in the next developments of our idea we extend our research in the direct interaction between human and drones. Starting from the problem of augmenting the teleoperation service with information coming from the cloud platform, we extend this idea towards a scenario where the drone has the possibility of augmenting with information the physical space. In particular adding to the drone the possibility of projecting or displaying those information in the human environment, so bypassing the teleoperation interface, and enable a direct interaction between human and drones.

5.1. Motivations

Nowadays we are witnessing a revolution in the ways of human interact with machine and digital information. The physical interfaces such as keyboard, mouse, tablets, smartphone and displays will be in the future not anymore the primary mean of interaction. Especially looking at the trends of the interfaces evolution, is clear that we are moving towards alternatives way of interact such as gestures and direct-touch interfaces (A. Erol, Oct. 2007) (A. D. Wilson, 2008). Very popular interfaces such as smartphone are currently very widespread but they need physical and visual attention, that is a big problem when the user is in a harsh mobile situation. Other emerging technologies such as Head Mounted Displays (HMD) are very promising but at the same time they could give an additional constraint and difficulties in the interaction to the user.

Augmented reality is a way to represent information in addition to the 3D environment but projected on a physical interface such as a HMD or hand-held device (Raskar, 2005). Spatial augmented (Raskar, 2005) reality is an extension of AR which exploit projectors to augment surfaces directly in the environment. The problem is that the environment needs to be instrumented with projectors that cover large areas with augmentations. So the idea is to extend spatial augmented reality with mobile robots.

Therefore we want to create an innovative Human Computer Interaction (HCI) combining robotics and spatial augmented reality for creating a novel way of interaction. In this scenario we envision small personal drones with on board pico- projectors and cameras for creating movable personal projections in the environment. So we call this new way of interaction *"User's Flying Organizer"* (UFO).

Therefore this device could be useful for supporting for example a technician during building maintenance. We imagine that a remote operator can support the technician projecting information directly in the environment, since the remote operator can supervise the technician or in addition also accessing information useful to display in the environment (such as building maps, hidden wires, plumbs, structural elements ecc..)

This scenario makes understand the advantages of having this device:

- The user doesn't need to wear any display such an HMD or a hand held display
- The field of view is potentially unlimited because the device is accompany the technician in the environment
- The input camera could be used for adding visual interaction such as gestures recognition

5.2. INTRODUCTION

Drones are rapidly becoming a commodity technology, even used for personal activities, e.g., aerial photography. We can easily imagine using inexpensive drones as part of our everyday tool set. Equipped with suitable sensors, such as cameras, these drones serve primarily as remote input devices. However, drones are rarely used for output. A display on a drone would have to be very small; mobile users prefer handheld or body-worn displays. Spatial augmented reality (Peters, 2005) overcomes display size constraints and avoids encumbering the user with mobile devices by projecting relevant information directly onto surfaces in the environment. However, existing spatial augmented reality systems are either stationary, or mobility is achieved by letting the user carry the projector. Obviously, the latter case fails in resolving the encumbering situation for the user. We propose a flying projector (Figure 32) as a new approach for encumbrance-free human-computer interaction. Our flying projector design can be used as a robotic companion, which follows the user and projects helpful information on relevant surfaces near the user. For instance, it can highlight the next machine part to attend to during a maintenance procedure. Building a flying projector requires mastering several challenges.

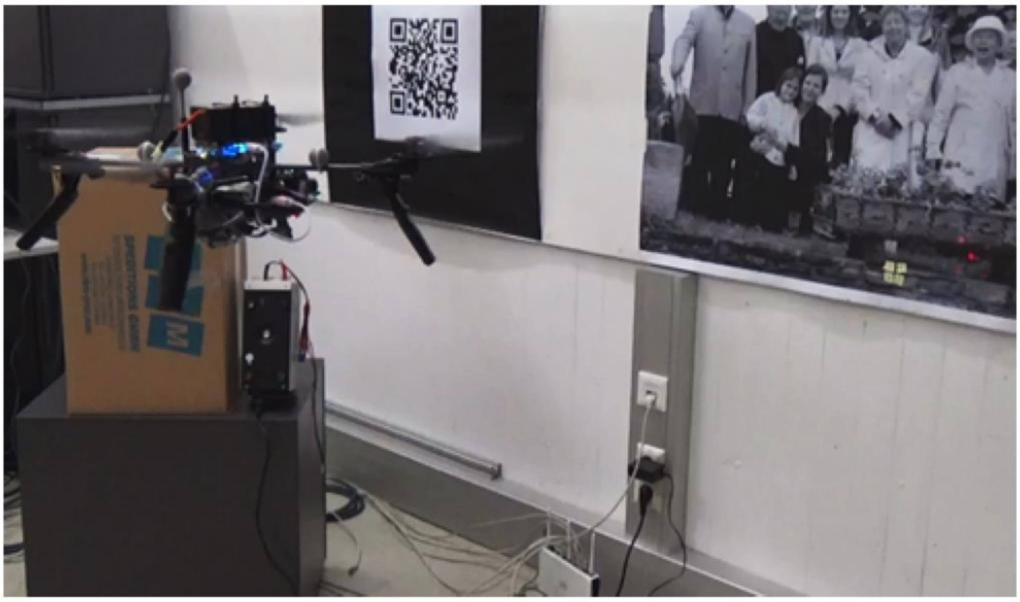


FIGURE 32 FLYING LASER PROJECTOR IN-FLIGHT.

First, safety considerations require that a drone operated as a companion near a person must be small and lightweight. This imposes significant restrictions concerning payload and power consumption, making it difficult to equip the drone with all required necessary input, output and compute units. To facilitate a flying projector, we introduce a novel lightweight laser projection system built from scratch. The projection system is integrated into a small scale microaerial vehicle (MAV). It is equipped with an auto-pilot to perform way-point following, and the projection system can project a set of basic symbols. Second, the quality of airborne projection is influenced by the stability of the MAV. Changes to position and orientation of the drone, while in hovering mode, as well as vibrations emerging from the rotors during flight must be considered. Thus, a stabilization method is necessary to compensate at least for significant movements of the drone. To this aim, we describe how a projector-camera feed-forward loop can compensate for pose fluctuations by deflecting the projector's laser beam. The feed-forward correction algorithm is based on pose estimates of a real-time natural feature tracker (NFT). We evaluate results of the projection stabilized by the poses of the feature tracker and compare it with the correction based on poses estimated by an Optitrack motion tracking system. As part of the proposed scenario, this chapter represents a first step towards combining the fields of robotics and spatial augmented reality. It focuses on the flying projection platform and presents the following contributions:

- a new lightweight laser projector,
- an implementation of a drone-based flying projector by integration of the laser projector onto a MAV
- an experimental evaluation of two methods for projection stabilization.

In this project, we share our experiences of building a new kind of human-robot interaction system. We provide details about what works and what does not work, which we believe to be interesting for researchers and engineers interested in a similar development.

5.3. Related work

A flying projector is a conceptual extension of a mobile projector. Since the early 2000s, spatial augmented reality has been a popular topic for mobile interaction. However, most approaches for room-sized environments either cover all relevant surfaces with arrays of projectors or rely on steerable projection. This idea was pioneered with the everywhere display projector (Pinhanez., 2001) and later extended with real time reconstruction from commodity depth sensors in the Beamatron (A. Wilson, 2012.). Other work considers mobile projectors, which are handheld or worn on the head or on the body. They can be combined with depth sensors (C. Harrison) or inertial sensors (K. Tajimi, 2010) to react to the user's movement and, potentially, a changing environment. However, visual light projectors with sufficient brightness for spatial interaction usually require a stationary power source. Truly mobile, battery-powered projectors can only operate at very short distances. Laser pointers concentrate the emitted energy in a single spot and, consequently, can achieve a significantly better contrast than conventional projectors with the same power budget. A steerable laser pointer can be used to point to a particular task location. A shoulder-mounted implementation of a steerable laser pointer has been used in a tele-assistance scenario (Ando., 2004). However, the same group of authors later proofs that a single point is not sufficient for conveying complex instructions (Sakata, 2006). Instead of a steerable single laser point, a scanning laser can be used. (Ando., 2004) describe a head-mounted projective display with a scanning laser mounted co-axial to the observer's eye. (B. Schwerdtfeger, 2008) explore head mounted and stationary scanning lasers for spatial augmentation. In both cases, the workspace is a major limitation. Because of the obvious technical difficulties, there has been little work on flying projectors. (J. Scheible, 2013) demonstrate outdoor flying projection with a commercial visual light projector weighting 200g, mounted on a large octocopter platform. These authors suggest a human-computer interaction scenario, but do not consider the problem of projection stabilization at all. The closest work to ours in terms of combining projection and drone flight was presented by (Y. Hosomizo, Oct 2014). They suggest a flying projection platform and approach the problem of image stabilization by combing dead reckoning and computer vision. However, they aim at a simpler solution than ours, using a commercial projection system and off board computation of the image stabilization. Specifically, they do not gather exact measurements of the drone's trajectory and the actual position of the projection. They also do not consider the distance to the projection surface during stabilization. Consequently, results for image stabilization are not evaluated in flight and only shown while the MAV is suspended from cables to meet weight constraints.

5.4. SYSTEM DESCRIPTION

In this paragraph, we describe our prototype and discuss its characteristics and limitations. The main components of our experimental system are the MAV itself, the on-board computer and the custom laser projector. In addition we use a motion tracking system combined with a ground station to control the MAV. An overview of the system is shown in Figure 33.

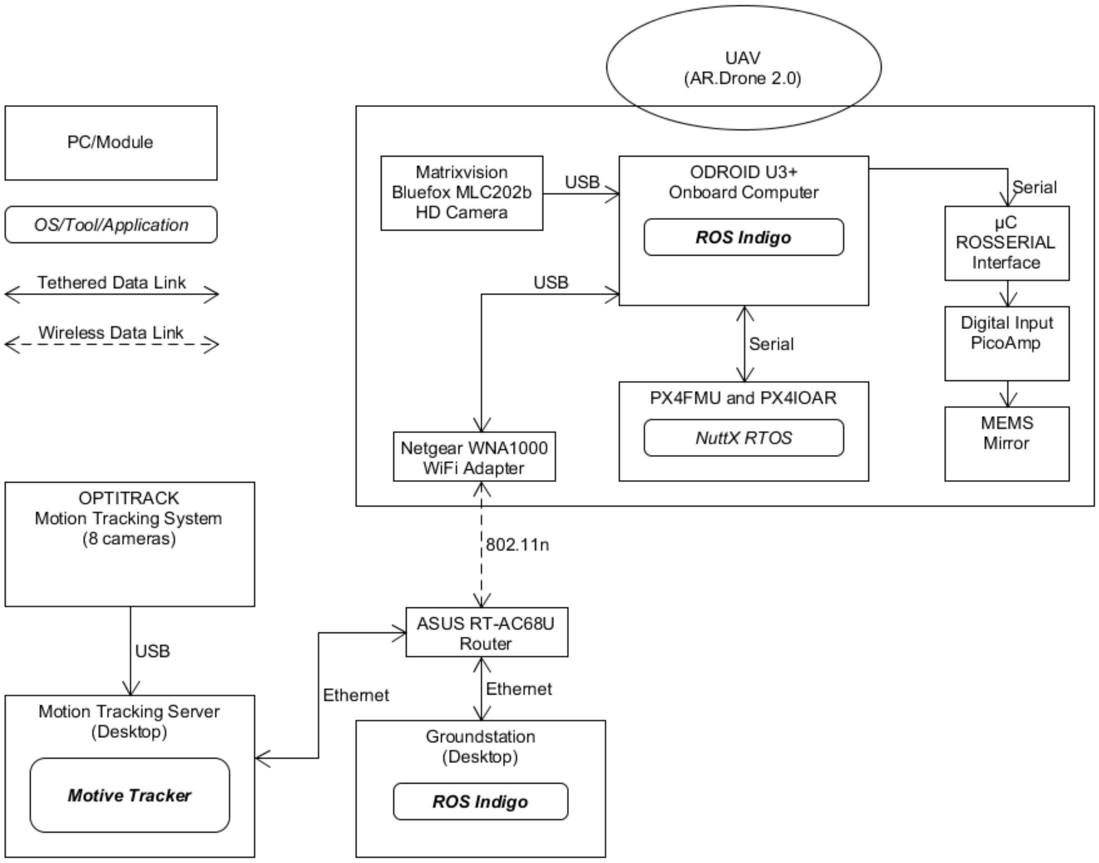


FIGURE 33 DIAGRAM OF THE EXPERIMENTAL SETUP. IT INCLUDES THE MOTION TRACKING SYSTEM WHICH IS BASED ON 8 TRACKING CAMERAS AND FURTHER CONNECTED TO THE GROUND STATION VIA ETHERNET. ADDITIONALLY THE GROUND STATION IS CONNECTED TO OUR MAV THROUGH WIFI LINK. THE LASER PROJECTION SYSTEM ITSELF IS MOUNTED ON THE MAV AND INTERFACED TO THE ONBOARD COMPUTER VIA SERIAL LINK.

In Figure 34, we show the flying MAV. The on-board computer and the components of the projection system are mounted below the flight management unit. The laser projector unit is mounted together with the camera in the front. The battery is mounted on top to achieve acceptable weight distribution.

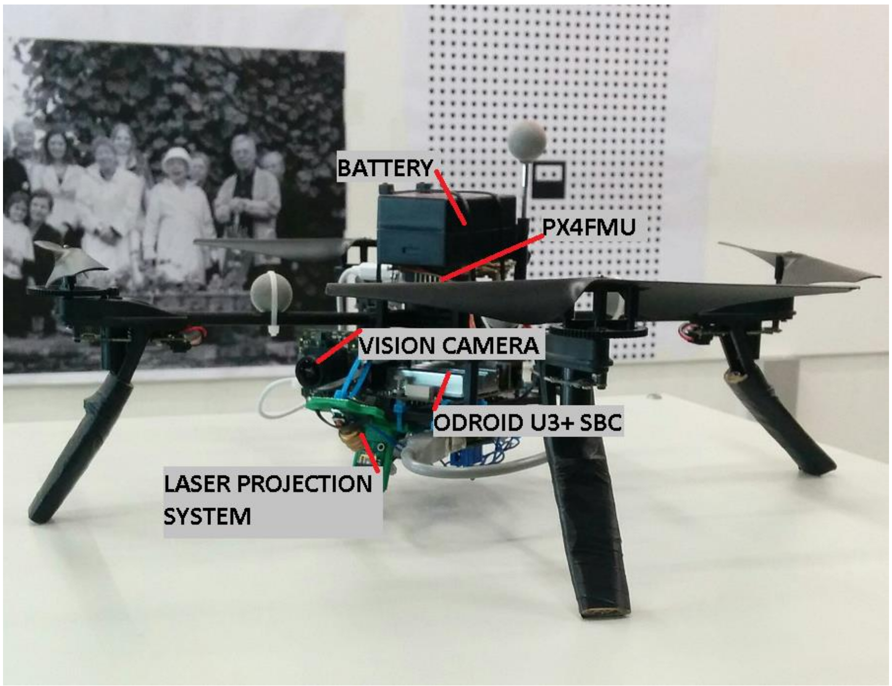


FIGURE 34 DRONE PLATFORM OF OUR EXPERIMENTAL SETUP INCLUDING THE MAIN COMPONENTS. THE BATTERY IS MOUNTED ON TOP FOR BALANCED WEIGHT DISTRIBUTION. BELOW THE BATTERY, THE PX4 LOW-LEVEL FLIGHT MANAGEMENT CONTROLLER AND THE PX4IOAR BOARD ARE LOCATED. BELOW THE FRAME, THE ON-BOARD COMPUTER AND THE LASER PROJECTION SYSTEM ARE POSITIONED. THE PROJECTION SYSTEM AND THE VISION CAMERA ARE FACING FORWARD.

5.4.1. Micro Aerial Vehicle

The MAV (55cm largest diameter, 600g weight) uses a semi-customized design with engines, rotors and frame taken from the ARDrone 2.0 platform. It is powered by a single 2000mAh battery with 11.1V. The flight time is limited to about 5 minutes, while running all relevant components and tasks for stabilization of the projection in flight. As a central processing unit, we use an ODROID U3+ single board computer with a Samsung Exynos4412 Prime Cortex-A9 quad-core CPU. We added a Pixhawk PX4 flight management controller (L. Meier, 2011) as a low-level flight control unit including attitude and pose estimation. The ODROID is connected to the PX4 via serial link and communicates with a ground-station via Wifi. It captures image data at 20Hz with 1280x960 resolution from a forward-looking MatrixVision Bluefox2 camera connected via USB 2.0. All high-level tasks like processing of the image data, pose estimation by visual feature tracker and sending feed-forward correction data to the laser projection system run on-board and are implemented in the ROS framework (M. Quigley, 2009).

5.4.2. LASER PROJECTION SYSTEM

We custom-built a laser projector (70g) from the following components: A 5mW laser module emits light at a wavelength of 650nm. The laser beam is deflected by a two-axis (tip/tilt) MEMS mirror (V. Milanovic, May 2004). Its reflective surface has 2DOF and can be steered in a range of ±6 ◦ by applying a bias differential driving scheme. To generate the required DC bias voltage, we use an amplifier interfaced to a microcontroller via SPI. The microcontroller is connected to the ODROID via a serial link, relying on the ROS-serial package. To preserve weight, no gimbal mounting was used. An overview of the laser projection platform is shown in Figure 35.

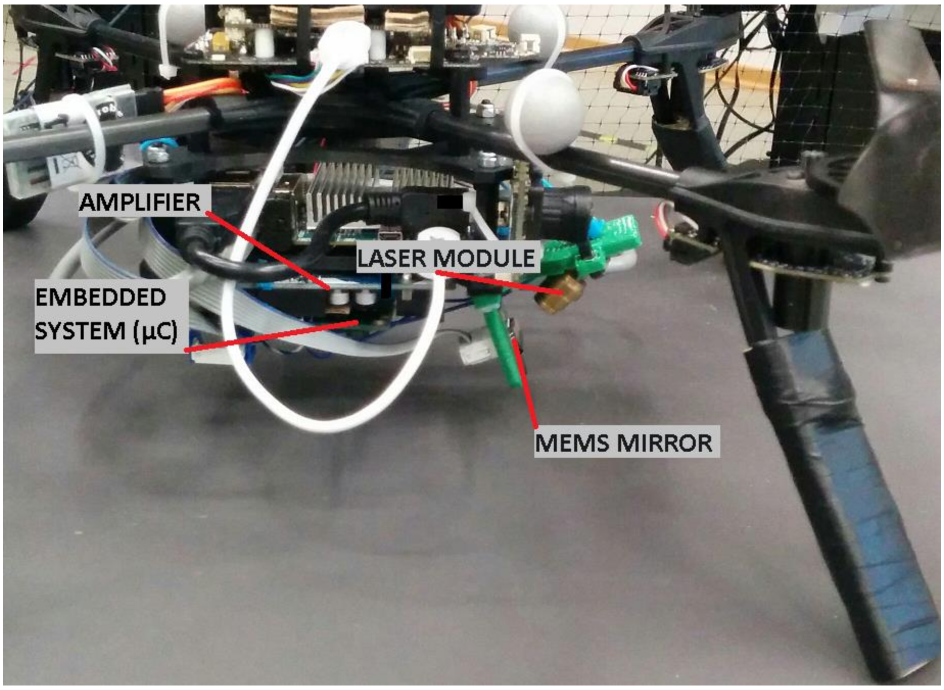


FIGURE 35 DETAILED OVERVIEW OF THE LASER PROJECTION SYSTEM. THE EMITTED LIGHT OF THE LASER MODULE IS REFLECTED BY A FORWARD FACING MEMS MIRROR. THE MIRROR ITSELF IS INTERFACED TO AN AMPLIFICATION STAGE, WHICH CONVERTS COMMANDS FROM THE EMBEDDED SYSTEM INTO APPROPRIATE VOLTAGES, STEERING THE MIRROR IN TWO DIRECTIONS

5.4.3. FLIGHT CONTROL OF THE MAV

For flight control of the MAV we use an Optitrack motion tracking system providing the PX4 flight management controller with low latency pose estimations at high rates. We use eight stationary cameras, covering an area of roughly 5 × 4 × 3m. Poses are delivered to the ODROID via Wifi at 120Hz, with 8ms latency. Time between the Optitrack ground-station and the PX4 is synchronized via NTP/Chrony (http://chrony.tuxfamily.org/).

5.4.4. POSE ESTIMATION FOR STABILIZATION OF PROJECTION

In addition we run a custom natural feature tracker onboard estimating poses based on the vision camera data. The natural feature tracker is similar to the work of (D. Wagner, May 2010). It detects visual features to compute the pose of the camera. In our setup, we use a known planar target with rich texture. This allows us to achieve a reasonably high frame rate and robust tracking of camera poses. Typically, the tracking takes about 30ms on the ODROID. It is remarkable that our experiments for projection stabilization rely on both systems the motion tracker and the feature tracker. Although for flight control of our MAV we use the motion tracking system only.

5.5. PROJECTION STABILIZATION

In this paragraph we want to concentrate on the problem of stabilizing projected information suffering from movements of the MAV. We propose a method for stabilization and describe our implementation deployed on the ODROID onboard computer.

5.5.1. COORDINATES FRAMES AND TRANSFORMATION

Fig. 36 shows the reference frames of our experimental setup: the world, the marker on the wall, the camera and the projector. Note that the camera is rotated by 90◦ around the z-axis. We stabilize the projected image using the 4 × 4 homogeneous transformation *Tpm* from projector to marker, consisting of the rigid transformations *Tpc* from projector to camera and the transformation *Tcm* from camera to marker, which is tracked by the natural feature tracker:

$$Tpm = TpcTcm$$

A point *Pm* in marker space can be transformed into projector space as $Pp = TpmPm$ and further on into projected space of the laser.

$$\begin{pmatrix} x \\ y \end{pmatrix} = K_p P_p = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} P_p$$

The projection matrix *Kp* of the laser is defined by the intrinsic parameters *fx*, *fy*, *cx* and *cy*; (x,y) are the pixel coordinates in projector space. Since we control the projector giving as input the two angles α β of the MEMS mirror, we need to create a function to transform from angles to pixels coordinates and vice versa. This transformation is done as follows:

$$u = Sx \cdot (\alpha - cx)\, v = Sy \cdot (\beta - cy)$$

where *cx, cy* are the coordinates of the principal point of the projector. Sx and Sy are two scale parameters depending on the maximum angle possible for the projector (in our case αmax = 6◦). The resolution of the projection is defined by $res = (resx, resy)$. It is then valid that

$$Sx = \alpha \max \left(\frac{resx}{2}\right)$$

$$Sy = -\alpha \max\left(\frac{resy}{2}\right)$$

By inverting the same function we get the transformation from projector pixel space into degree inputs of the laser projection interface.

5.5.2. Camera-Projector Calibration

In our preliminary experiments, we had to manually calibrate camera-projector intrinsic and extrinsic parameters. This is because we found the laser-projector distortion behaves different from a pinhole camera model (see Fig. 36). We experimented with camera-projector calibration using a combined pinhole and radial distortion model for the laser projector. However, the calibration did not converge to reasonable results. Forcing the projector to follow a linear pinhole model results in more reasonable results with a re-projection error around 21 pixels. We believe the non-linear distortion effect of the laser projector is a limitation of the current hardware setup. Therefore, we manually calibrate parameters to fit linear pinhole camera model for the laser projector. This results in acceptable calibration for our experiments. Fig. 37 shows some of the captured images used for intrinsics and extrinsics camera-projector calibration. Note the back-projected laser dot (green cross) after calibration. As shown, the re-projection errors of laser dots are quite high. As discussed previously, such high errors likely come from low accuracy of the MEMS mirror used for laser projection. In the calibration setup, we used a 7x6 dots calibration pattern and 25 laser dots projection where laser dots locations are uniformly distributed throughout the laser projector pixel coordinates. We assume that the camera intrisics are precalibrated. This allows us to estimate 6DOF relative transformation between camera and laser projector, and intrinsics of the projector. The estimation is done similar to (Falcao).

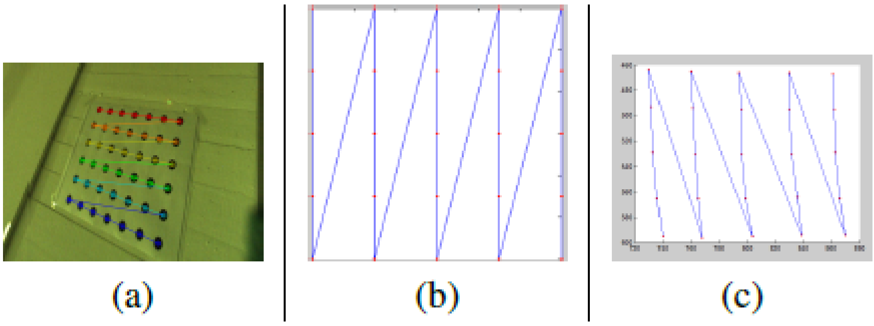


FIGURE 36 FIG. (A) ILLUSTRATES DETECTED LASER DOT (GREEN CROSS) AND DETECTED CALIBRATION PATTERN IN OF THE CAMERA CAPTURED IMAGE USED TO FOR CAMERAPROJECTOR CALIBRATION. FIG. (B) SHOWS PIXEL LOCATIONS OF PROJECTING POINTS IN PROJECTOR PIXEL COORDINATES. FIG. (C) SHOWS DETECTED LASER DOTS IN CAMERA

IMAGE WHEN THE LASER DOTS ARE PROJECTED ON A PLANAR WALL. THE STRANGE DISTORTION IN (C) SUGGESTS THAT OUR LASER-PROJECTOR DOES NOT FOLLOW A PINHOLE CAMERA MODEL WITH RADIAL DISTORTION, AS COMMONLY USED FOR CAMERA MODEL.

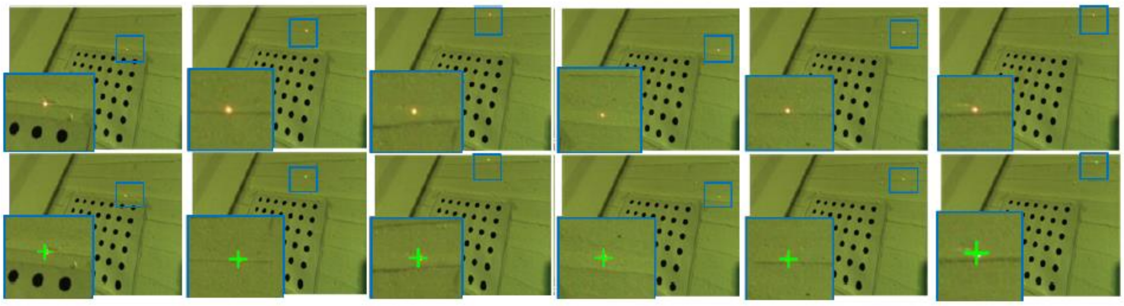


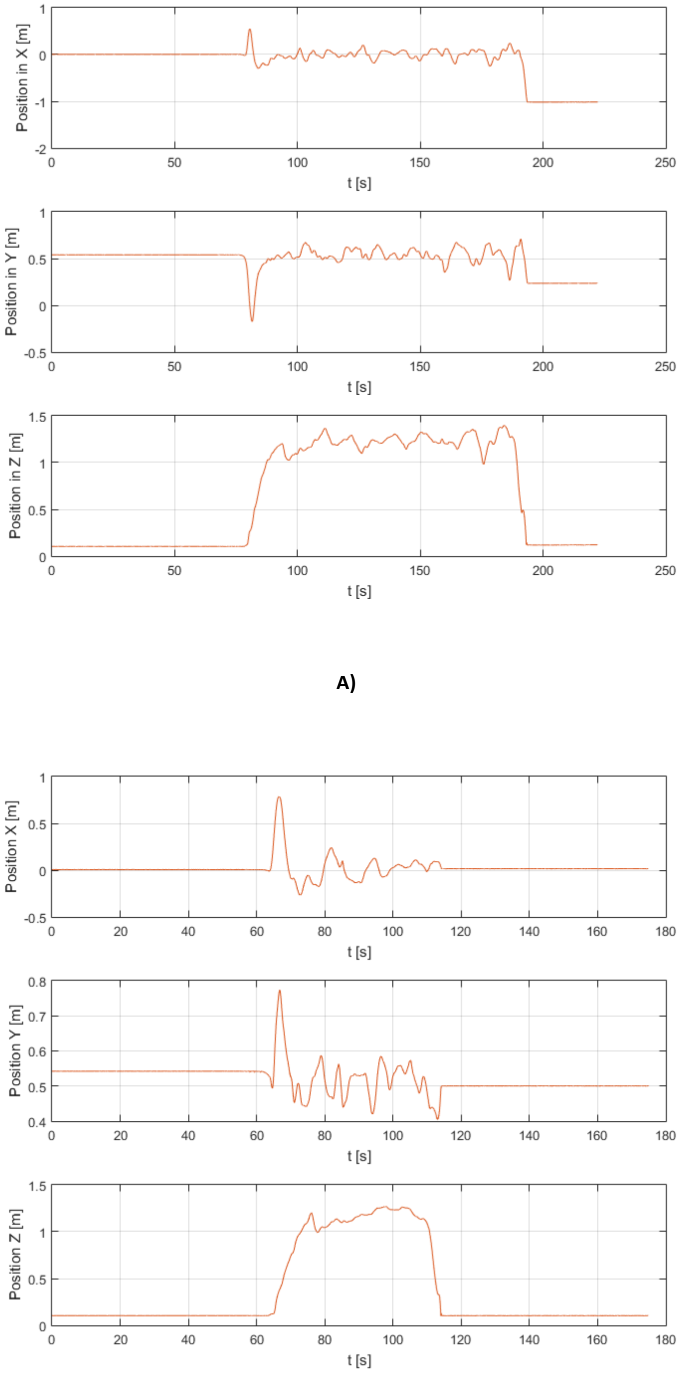
FIGURE 37 THESE ARE SOME OF CAPTURED IMAGES USED FOR CAMERA-PROJECTOR CALIBRATION EXPRIMENTS. THE TOP ROW CONTAINS INPUT IMAGES WITH PROJECTED LASER DOTS IN THE CAMERA FIELD-OF-VIEW. THE BOTTOM ROW SHOWS VALIDATION RESULTS WHERE ESTIMATED CAMERA-PROJECTOR INTRINSICS AND EXTRINSICS ARE USED TO TRANSFORM AND RE-PROJECT LASER DOTS TO CAMERA IMAGE. THE RE-PROJECTED LASER DOTS ARE HIGHLIGHTED WITH GREEN CROSSES.

5.6. COMPENSATION VIA FEED-FORWARD

For stabilization of the projected information, we want to use a simple, but robust feed forward correction algorithm. We choose an arbitrary 3D point *Pm* in the marker frame *Fmarker* and transform it into camera coordinate frame *Fcam* using the pose of the NFT. We then use camera to projector transformation to derive coordinates of the defined point in the laser projector frame *Fproj* . Finally we steer the mirror to the updated position of the defined 3D arbitrary point. The tracked marker is mounted on a wall and trackable from a distance of up to 3m. The images with a resolution of 1280x960 are delivered to the on-board computer with a latency of 200ms. An additional 30ms have to be taken into consideration for the natural feature tracker. The estimated pose is used to transform the output coordinates into the projected frame of the laser. For correction, the updated coordinates in laser space are finally forwarded to the laser projection interface. The latency of the final step is approximately 20ms

5.7. EXPERIMENTS

In this section, we analyze the performance of the stabilized projection during flight. We report on two experiments: First, we elaborate on the accuracy of feed-forward stabilization based on poses derived from the natural feature tracker. Second, we compare the results with compensation computed from Optitrack data. In a first step we position the MAV in a static pose. In the next step we project the point (0,0) in projector coordinates onto the marker. We then measure this 3D point in marker coordinates and define it as our desired arbitrary point. We also capture this pose of the drone and define it as our desired waypoint for hover flight in our experiment. In the experiment we place the MAV randomly in a flying space covered by the motion tracking system and lift off. We then command the MAV to approach the predefined waypoint and keep it in hovering position. During hover flight the pose of the wall mounted marker is localized. We then use our onboard feedforward correction algorithm to project the desired point on the marker. At the same time we project the uncompensated point (0,0) in projector coordinates which represents the disturbance due to the movements of the MAV. To quantify the accuracy of the compensation, we record position and altitude of the MAV (Figure 38 a and Figure 38 b) and capture image data over 20s to measure the projected points. Figure 39 compares the measured trajectory of the first experiment whereby the compensated point with the uncompensated center of projection is shown. Figure 40 shows the results using Optitrack for stabilizing the image.



A)

B)

FIGURE 38 (A-B) BOTH FIGURES SHOW THE TRAJECTORY OF THE MAV MEASURED WITH OPTITRACK. FIG. (A) SHOWS MAV TRAJECTORY OF FIRST EXPERIMENT (NFT). FIG. (B) SHOWS SAME DATA BUT FOR THE SECOND EXPERIMENT (COMPENSATION BASED ON OPTITRACK POSES). FLIGHT TIME OF NFT IS INCREASED BECAUSE OF DELAYED TRACKING OF MARKER.

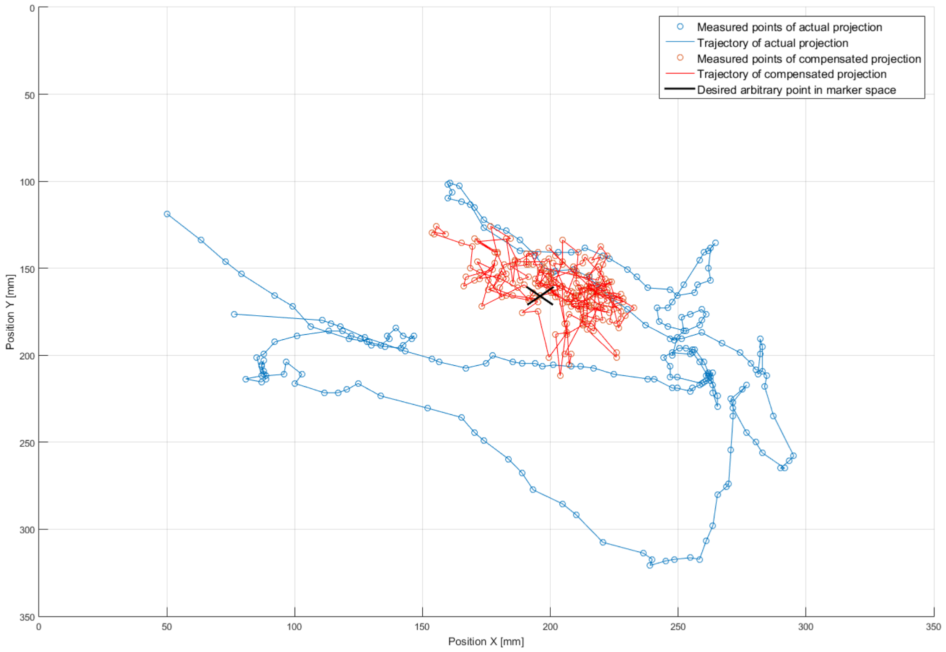
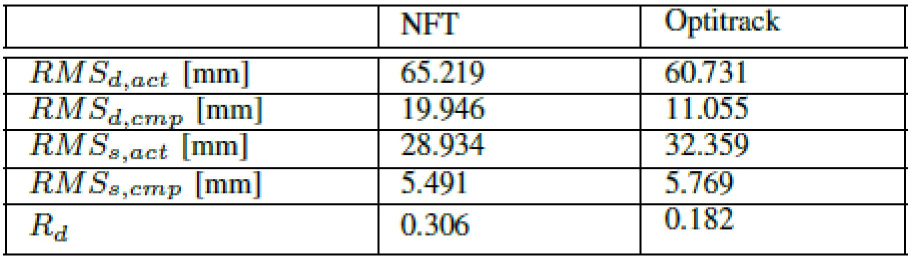


FIGURE 39 TRAJECTORIES OF ACTUAL AND COMPENSATED PROJECTION, BASED ON NFT POSES. THE BLACK CROSS REPRESENTS THE DESIRED ARBITRARY POINT IN MARKER SPACE.



	NFT	Optitrack
$RMS_{d,act}$ [mm]	65.219	60.731
$RMS_{d,cmp}$ [mm]	19.946	11.055
$RMS_{s,act}$ [mm]	28.934	32.359
$RMS_{s,cmp}$ [mm]	5.491	5.769
R_d	0.306	0.182

TABLE 4 ERROR CHARACTERISTIC AND ERROR COMPENSATION

It can be clearly seen that the dynamic error is considerably reduced by the feed-forward correction. Due to the much lower latency, the RMS error is significantly improved with the Optitrack. We additionally take a set of measurements positioning the MAV statically in front of the marker to inspect on the static error. At different poses close to the defined waypoint, which we also use for inflight projection, we then measure the same RMS error characteristics as noted in terms of dynamic error. The dynamic error characteristics are thereby shown in Table I. The RMS values of the dynamic error without compensation are denoted as RMSd,act (distance from desired to actual point in marker space). The dynamic error values measured from the compensated points are denoted as RMSd,cmp (distance from desired to compensated point in marker space). RMSs,act and RMSs,cmp represent characteristics of the static error accordingly. We further define an error reduction ratio Rd for the dynamic error, which is the relation between the error of the compensated points and the error of the uncompensated points.

$$Rd \ = \frac{RMS_{d,cmp}}{RMS_{d,act}}$$

5.8. DISCUSSION AND CONCLUSIONS

Evaluation of our system reflects current limitations related to hardware constraints. In particular limits of the power-supply and payload for drones of this size are the main reason for reduced flight time while computational power is still hardly sufficient. However, facing those limitations, we have presented a novel platform which is capable of dynamic flight and projection at the same time. This is made possible by a custom design utilizing a small sized MAV. We achieve a lightweight and low-power flying projection system stabilizing projected images in 3D environment and safe to operate near human. In a future version we want to improve overall performance of the flying projector. This would include to even more decrease weight of the projection system and payload in general. Additionally, we would like to further investigate the distortion effect of the laser projector as we suppose that such distortion is mainly owed to an inaccurate surface of the MEMS mirror. Finally we want to use more powerful methods in terms of pose estimation and error correction to make the platform even more flexible inside of future human machine interaction scenarios.

6. PROJECTOR CALIBRATION

In previous chapter there was an explanation of the idea of having a flying projector and an outline of what are the main problems of applying this technology in reality. In particular we can say that although there are still many technology constraints the overall result is satisfactory. One of the most important requirements that we needed to fulfill is to apply a more reliable calibration regarding the camera to projector calibration. In the previous chapter we only performed a manual calibration because we couldn't get a satisfactory result. That is mainly due to the fact that the laser projector we used in this project is very particular, since we were looking for the best hardware according to the lowest weight solution. We used a MEMS mirror (Figure 40) to reflect the laser beam and for steering the mirror we used a driver (Figure 41) from Mirrocle.

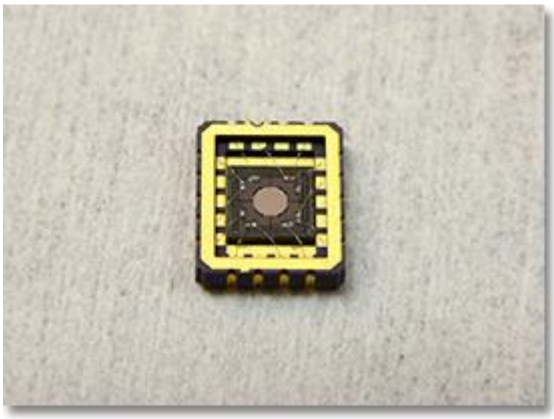


FIGURE 40 MEMS MIRROR

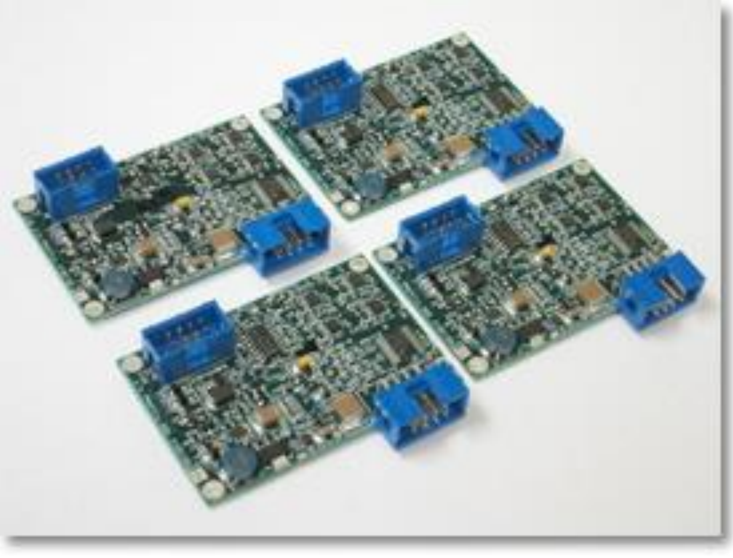


FIGURE 41 CONTROL BOARD FOR THE MIRROR MEMS

As we described in paragraph 5.4.2. we have custom built laser projector:

- 70 grams of weight
- 5mW laser module to emit the laser beam at a wavelength of 650nm
- MEMS angle range is ±6◦
- The control board for steering the laser beam has a SPI interface
- The control board communicates with Arduino Uno with a custom driver written by us (https://www.arduino.cc/en/Main/ArduinoBoardUno)
- Arduino communicates then with Odroid U3 exploiting the ROS Serial package (http://wiki.ros.org/rosserial)

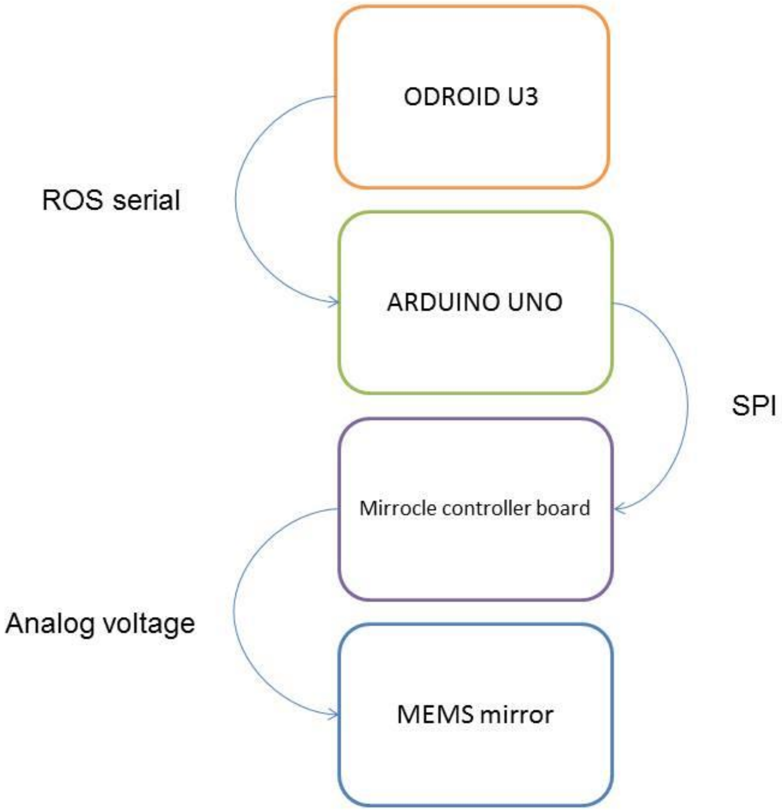


FIGURE 42 PROJECTOR SET UP AND DEVICES

So with this set up we performed a camera to projector calibration composed by two parts:

- Extrinsic calibration: estimate the pose (rotation and translation) from the camera to the projector
- Intrinsic calibration: the deformation of the laser mirror

In this work we start with the extrinsic calibration since we believe it is more important for improving the result in the final application.

6.1. Extrinsic calibration

As already described in this chapter we are focusing more the extrinsic calibration. The novelty of this calibration is mainly due to the fact that the projector we are using is very new and there was no previous knowledge about that. In addition we needed to find a way to calibrate the projector bypassing completely the intrinsic model of the laser mirror since that was completely unknown. So in this chapter there is an explanation of the theoretical problem and how we solved it.

Let's start to define the problem. We put a reference (right-handed) for the camera system, the projector and the marker where we will project the central point, with coordinates (0,0), of the laser projector. We call respectively $T_{CameraMarker}$ the transformation (rotations and translations) that brings the reference frame from the Marker to the Camera. $T_{ProjectorMarker}$ the transformation from marker space to projector space. And $T_{ProjectorCamera}$ the transformation from camera to projector that is the unknown that has to be calculated by the calibration. We calibrate the projector only projecting the central point in order to bypass the intrinsic parameter of the laser projector and then optimizing for every pose.

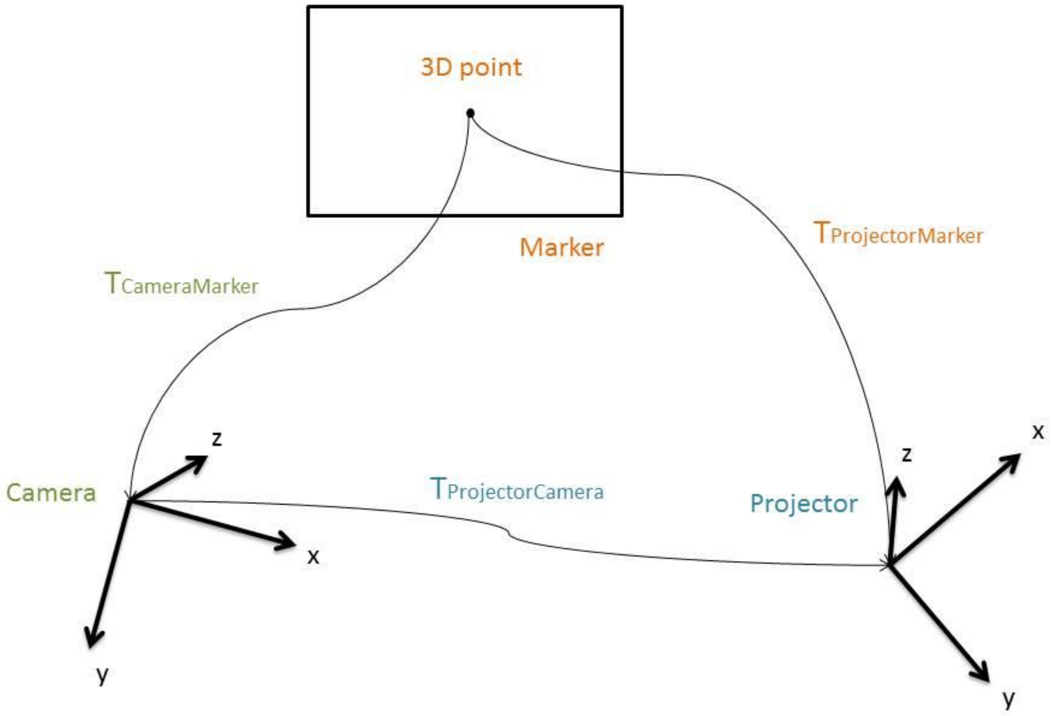


FIGURE 43 CAMERA PROJECTOR CALIBRATION

We took 40 poses $T_{CameraMarker}$ with the Natural Feature Detection (Look3D) developed in the Institute of Computer Graphics and Vision of the Technical University of Graz.

We wrote the transformation matrix as follows:

$$T_{CameraMarker} = (R_{CameraMarker}|T_{CameraMarker})$$

$$= \left(\begin{array}{ccc|c} rcm11 & rcm12 & rcm13 & tcm1 \\ rcm21 & rcm22 & rcm23 & tcm2 \\ rcm31 & rcm32 & rcm33 & tcm3 \\ & 0\ \ 0\ \ 0 & & 1 \end{array}\right)$$

$$T_{ProjectorMarker} = (R_{ProjectorMarker}|T_{ProjectorMarker})$$

$$= \left(\begin{array}{ccc|c} rpm11 & rpm12 & rpm13 & tpm1 \\ rpm21 & rpm22 & rpm23 & tpm2 \\ rpm31 & rpm32 & rpm33 & tpm3 \\ & 0\ \ 0\ \ 0 & & 1 \end{array}\right)$$

$$T_{ProjectorCamera} = (R_{ProjectorCamera}|T_{ProjectorCamera})$$

$$= \left(\begin{array}{ccc|c} rpc11 & rpc12 & rpc13 & tpc1 \\ rpc21 & rpc22 & rpc23 & tpc2 \\ rpc31 & rpc32 & rpc33 & tpc3 \\ & 0\ \ 0\ \ 0 & & 1 \end{array}\right)$$

So projecting always the central point from the projector we have the equations:

$$P_{projector} = T_{ProjectorMarker} * P_{Marker}$$

$$P_{projector} = T_{ProjectorCamera}T_{CameraMarker} * P_{Marker}$$

$$P_{projector} = \begin{pmatrix} X_{projector} \\ Y_{projector} \\ Z_{projector} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ Z_{projector} \\ 1 \end{pmatrix}$$

$$P_{marker} = \begin{pmatrix} X_{marker} \\ Y_{marker} \\ Z_{marker} \\ 1 \end{pmatrix}$$

Where the pose $T_{ProjectorCamera}$ is unknown, the P_{Marker} are the 3D laser points visible on the marker attached to the wall, the $P_{projector}$ are the points back projected from the 3D points in the environment in the projector space.

So the general equation for every of the 40 poses appear:

$$\begin{pmatrix} 0 \\ 0 \\ Z_{projector} \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} rpc11 & rpc12 & rpc13 & tpc1 \\ rpc21 & rpc22 & rpc23 & tpc2 \\ rpc31 & rpc32 & rpc33 & tpc3 \\ \multicolumn{3}{c|}{0 \quad 0 \quad 0} & 1 \end{array}\right) \left(\begin{array}{ccc|c} rcm11 & rcm12 & rcm13 & tcm1 \\ rcm21 & rcm22 & rcm23 & tcm2 \\ rcm31 & rcm32 & rcm33 & tcm3 \\ \multicolumn{3}{c|}{0 \quad 0 \quad 0} & 1 \end{array}\right) \begin{pmatrix} X_{marker} \\ Y_{marker} \\ Z_{marker} \\ 1 \end{pmatrix}$$

where P_{marker} is taken according to its pose $T_{CameraMarker}$ and the $T_{ProjectorCamera}$ remains the same and has to be computed.

So writing this equation for every of the 40 poses we can compute the Jacobian and optimize with the gradient descendent method for getting the solution.

6.2. First experiment – camera to projector calibration

As a first experiment we took 50 poses $T_{CameraMarker}$ estimated with the Natural Features Tracking (Look3D - https://github.com/sgabello1/Look3DROS) while projecting the central point, (0,0) point in projector space, from the projector on a chess board attached on the lab wall.

For every $T_{CameraMarker}$ pose we tried to hit with the projected point one corner of the chess board for achieving a precise measure of the 3D point projected, since we know the exact dimension of the chess board.

Then we optimized with gradient descendent. We initialize the first guess of the optimization with the manual calibration result.

We organize the vector of unknowns as:

$$x = \begin{bmatrix} Yaw \\ Pitch \\ Roll \\ Translation_{overX} \\ Translation_{overY} \\ Translation_{overZ} \end{bmatrix}$$

and the initial guess taken from the manual calibration was:

$$x_0 \;=\; [-90.0;\; 2.5;\; -9.8;\; 0.04;\; 0.02;\; 0];$$

Angles roll, pitch and yaw are in degrees and translations in meters.

So apply the optimization with 20 iterations, computing the Jacobian with a step of 10^-17 stopping the optimization with an error smaller than 10^-4 we got:

$$x \;=\; [\,-90,\;\; 2.8722,\quad -9.8748\,,\quad 0.0127,\quad 0.0127,\quad -0.0175];$$

At the end of the optimization we can say that, although it was quite difficult to take those 50 poses and measuring correctly the projected 3D points, the result was quite satisfactory. The main problem we encountered was that the magnitude of our measurement was really close to the magnitude of the noise so it was quite hard to optimize. Then looking at the angles values after the optimization result we can say that they are not changed too much from the initial guess so that probably means that somehow the initial guess was quite precise. A completely different behavior we can notice in the translation parameters that, although they should be easier to measure, they wasn’t and especially the Z coordinate was not very precise probably because it was quite hard to understand where the exact plan of reflection of the laser mirror was. That is again due to the fact that our laser projector is a very particular one.

6.3. Second experiment – drone to projector calibration (Optitrack)

In this second experiment we wanted to calibrate not anymore the projector according to the camera but to the drone frame. That because in the previous experiments (Chapter 5) we had many delays with the camera tracking and we wanted to test the flying projector at the best of its performances. Therefore we do the same transformation chain, but this time is from the drone frame instead of the camera frame. So our equations substantially remain the same but we only change the name of the transformations, in particular we have:

$$T_{camera} = T_{drone}$$

$$T_{marker} \ = \ T_{world}$$

$$T_{projector} \ = \ T_{projector}$$

Then we have another important improvement that we measure these poses and the 3D points with Optitrack just putting some AR markers where we want to get the measure, getting a very precise measure.

The transformation chain so will be:

$$P_{drone} = T_{DroneProjector} * P_{projector}$$

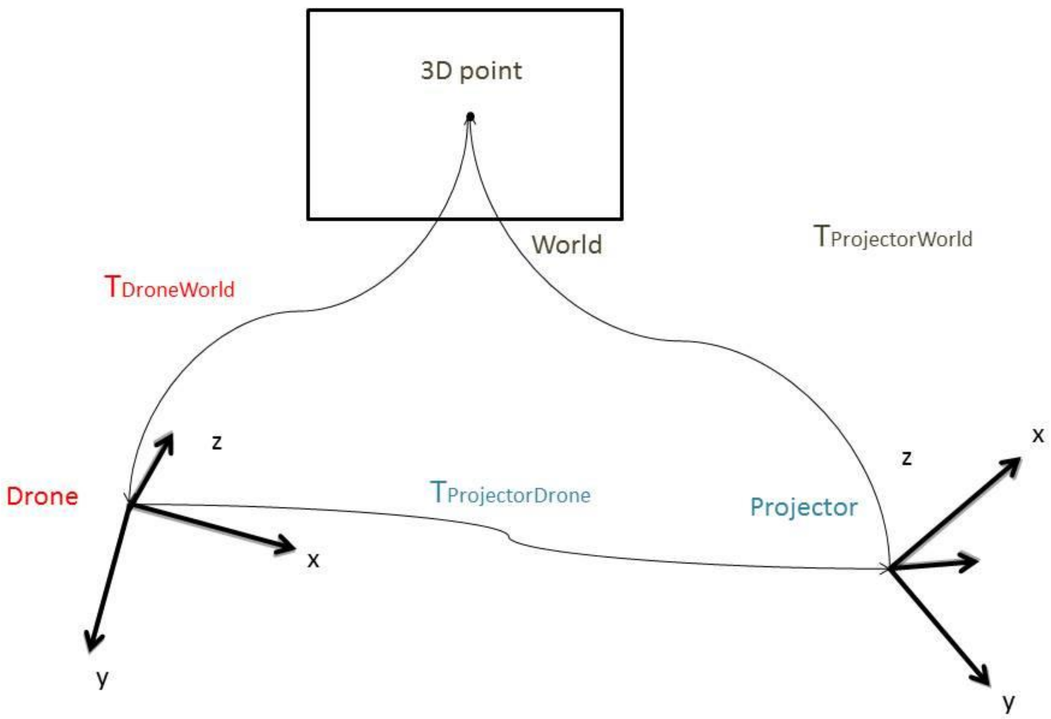


FIGURE 44 TRANSFORMATION CHAIN

Where P_{drone} and $P_{projector}$ are the points respectively in drone space and projector space. Then the transformation remain the same as depicted in the previous picture.

Then as in the previous experiment we optimized starting with the first initial guess with the manual calibration.

$$x_0 \ = \ [-1.776; \ \ -16.226; \ 0.0; \ -0.003164 \, ; \ 0.0368970; \ \ 0.02080]$$

Then optimizing with 13 steps we stop with an error of 10^-7 and a step of 2*10^-8 for computing the Jacobian we got the result:

$$x \ = \ [\, -1.8039, \qquad -16.2260, \qquad 0.0105, -0.0032, 0.0369, \qquad 0.0208 \,]$$

We report in the following the decreasing error during the optimization.

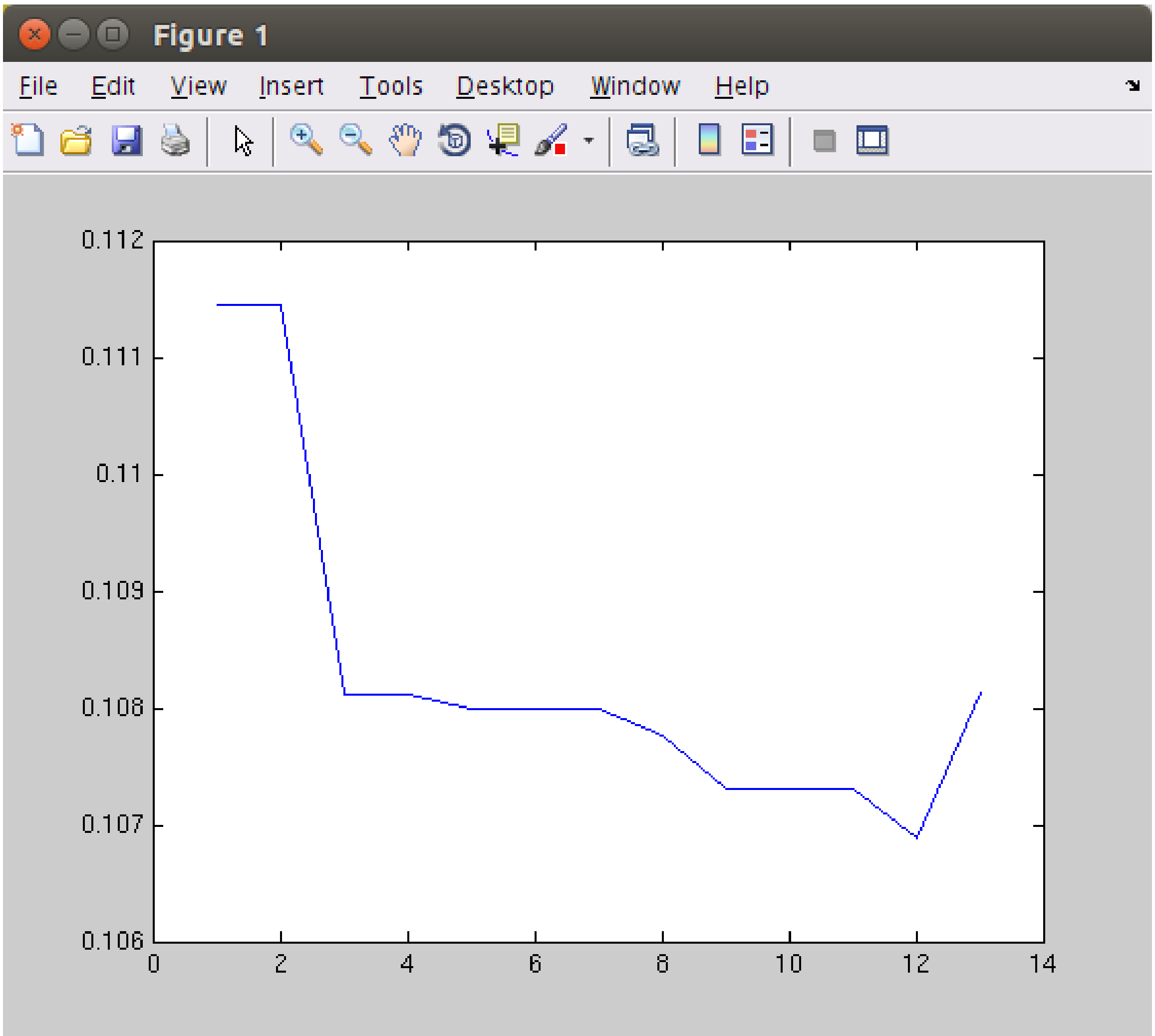


FIGURE 45 CALIBRATION RESULT

6.4. CONCLUSION AND DISCUSSION

In this chapter there is a proposed way to calibrate the laser projector according to the camera and the drone frame exploiting in this case Optitrack tracking system. As the experiments demonstrates we can assume the result from this calibration could be satisfactory.

It has to be noticed that we wrote the problem with non-linear equations that is actually harder than writing the problem with linear equations since an optimization is necessary and the final result is only estimated while is not computed exactly like in the linear case. That is due to the fact that our first attempt was to write the problem with linear equations, so having as unknowns 9 rotations parameters and 3 translations:

$$x_{param} = [r_{pc}11, r_{pc}12, r_{pc}13, r_{pc}21, r_{pc}22, r_{pc}23, r_{pc}31, r_{pc}32, r_{pc}33, t_{pc}1, t_{pc}2, t_{pc}3\ ]^T$$

Then writing the problem as before:

$$\begin{pmatrix} X_{projector} \\ Y_{projector} \\ Z_{projector} \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} rpc11 & rpc12 & rpc13 & tpc1 \\ rpc21 & rpc22 & rpc23 & tpc2 \\ rpc31 & rpc32 & rpc33 & tpc3 \\ & 0\quad 0\quad 0 & & 1 \end{array}\right) \left(\begin{array}{ccc|c} rcm11 & rcm12 & rcm13 & tcm1 \\ rcm21 & rcm22 & rcm23 & tcm2 \\ rcm31 & rcm32 & rcm33 & tcm3 \\ & 0\quad 0\quad 0 & & 1 \end{array}\right) \begin{pmatrix} X_{marker} \\ Y_{marker} \\ Z_{marker} \\ 1 \end{pmatrix}$$

But this time re-writing the equations in order to extract the 12 unknowns, taking into account that we are again back projecting the (0,0) central point of the laser projector so $X_{projector} = Y_{projector} = 0$:

$$0 = A * [r_{pc}11, r_{pc}12, r_{pc}13, r_{pc}21, r_{pc}22, r_{pc}23, r_{pc}31, r_{pc}32, r_{pc}33, t_{pc}1, t_{pc}2, t_{pc}3\ ]^T$$

Where matrix A is computed re-arranging the previous equations. Writing in this way the equations we have that for every point we have 2 equations. Therefore for solving the system we need at least 6 points.

Then to better understand why this approach doesn't work in our case we need to write the equations for one point. We would get something similar to:

$$\begin{bmatrix} -X_{marker} & 0 \\ -Y_{marker} & 0 \\ -Z_{marker} & 0 \\ -1 & 0 \\ 0 & -X_{marker} \\ 0 & -Y_{marker} \\ 0 & -Z_{marker} \\ 0 & -1 \\ X_{projector} * X_{marker} & Y_{projector} * X_{marker} \\ X_{projector} * Y_{marker} & Y_{projector} * Y_{marker} \\ X_{projector} * Z_{marker} & Y_{projector} * Z_{marker} \\ X_{projector} & Y_{projector} \end{bmatrix}^T * x_{param} = 0$$

So written in this form is clear that if we are projecting the (0,0) point from the projector it means that $X_{projector} = Y_{projector} = 0$ that makes the previous equation appear like:

$$\begin{bmatrix} -X_{marker} & 0 \\ -Y_{marker} & 0 \\ -Z_{marker} & 0 \\ -1 & 0 \\ 0 & -X_{marker} \\ 0 & -Y_{marker} \\ 0 & -Z_{marker} \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}^T * x_{param} = 0$$

Therefore the system becomes non solvable since we lose the information of 4 parameters for every equation.

CONCLUSION AND DISCUSSION

In this thesis are presented different ways of interaction thought the explanation of practical projects. The order of the chapters represents the historical order of the projects in which I was involved during my PhD. In the first chapter there is the state of the art of Human to Drone interaction. Starting from the definition of UAV, then the current HMI interfaces and most recent advancements in interaction techniques with drones.

In the second a brief explanation of the cloud robotics architecture and the main functionalities. In the third chapter the project Fly For Smart City, where is the vision of a smart city surveilled by drones and internet services in the cloud robotics platform. In this project there is also the innovation of exploiting Open Data accessible from the Internet for managing the mission of a drone. In the fourth chapter the extension of Fly For Smart Coty project with a *sliding autonomy* approach and the definition of different degrees of autonomy. In the fifth chapter the UFO project, where state of the art Spatial Augmented Reality technologies are applied to UAVs for augmenting the surrounding environment with information. The last chapter is the technical solution to a problem that came out in the previous chapter about camera to projector calibration. This last project was in collaboration with the Institute of Computer Graphics of the Technical University of Graz and is currently an on-going project (in the picture below the next improvement as a flying teaching assistant that projects information to a student).

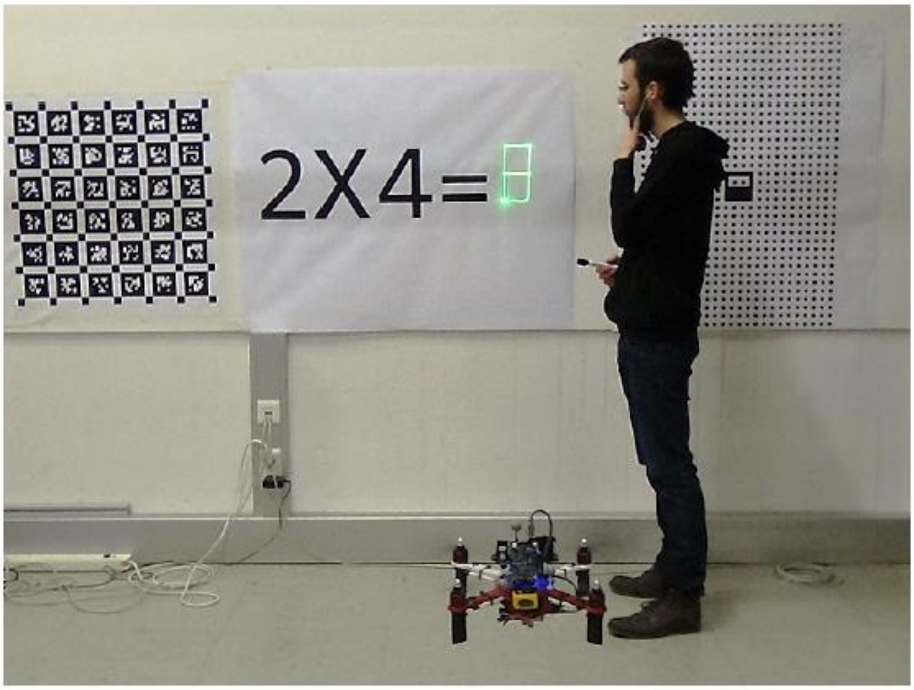


FIGURE 46 FLYING TEACHING ASSISTANT

APPENDIX

FLY 4SMARTCITY PLATFORM MESSAGES

MISSION PLANNER MESSAGES SET

LISTING 1: ACTION.MSG

```
string name

int8 DEVICE = 0

int8 FEEDBACK = 1

int8 CHECK = 2

int8 UNICAST_NOTIFY = 3

int8 MULTICAST_NOTIFY = 4

int8 BROADCAST_NOTIFY = 5

# enumerate above

int8 type

# filled only in case of DeviceAction and Feedback

string action_name

# filled only in case of DeviceAction and Feedback

Parameter[] parameters

# filled only in case of Notify (any kind) or CheckNotification

string slot_name

# filled only in case of MulticastNotify or UnicastNotify

string[] receivers_name
```

LISTING 2: BOOL.MSG

```
Bool data
```

LISTING 3: COORDINATE.MSG

```
float64 latitude

float64 longitude

float64 altitude

float64 heading
```

LISTING 4: COORDINATEARRAY.MSG

```
Coordinate[] waypoint
```

LISTING 5: DRONE.MSG

```
Header header
string name
string type_name
Coordinate home
Move[] movements
string[] slot_names
uint8 SAFE=0
uint8 NORMAL=1
uint8 AGGRESSIVE=2
uint8 travel_mode
```

Listing 6: ListString.msg

```
std_msgs/String[] list
```

Listing 7: Move.msg

```
string name
uint8 START=0
uint8 STOP=1
uint8 TAKE_OFF=2
uint8 LAND=3
uint8 GO_TO=4
uint8 HOVER=5
```

```
uint8 CIRCLE=6
uint8 HEAD_TO=7
#enumerated above
uint8 type
Action[] pre_actions
Action[] post_actions
#filled only in case of TakeOff and Circle moves
float64 altitude
#filled only in case of Goto or Circle move
Coordinate target_position
uint8 DIRECT=0
uint8 HORIZONTAL_FIRST=1
uint8 VERTICAL_FIRST=2
#enumerated above, filled only in case of GoTo move
uint8 strategy
#filled only in case of Hover or Circle moves
duration duration
#following parameters are filled only in case Circle move
float64 radius
bool clockwise
#filled only in case of HeadTo move
float64 direction
```

Listing 8: Parameter.msg

```
string key
string value
```

Listing 9: SensorPacket.msg

```
# GPS_current_position
float64 c_longit
float64 c_latit
int32 c_altit
# GPS_home_position
float64 h_longit
float64 h_latit
int32 h_altit
# GPS_target_position
float64 t_longit
float64 t_latit
int32 t_altit
# Flight_status
uint8 current_wayp
uint8 tot_wayp
int16 altimeter
```

OPEN DATA MESSAGES SET

Listing 10: Coordinates.msg

```
float64 x
float64 y
float64 z
```

Listing 11: Data.msg

```
Status status

Open_data[] data
```

Listing 12: OpenData.msg

```
int8 TYPE_STATIC=0
uint8 TYPE_DYNAMIC=1
uint8 type
string label
Parameter[] attributes
geometry_msgs/Polygon area
```

Listing 13: Parameters.msg

```
string key
string value
```

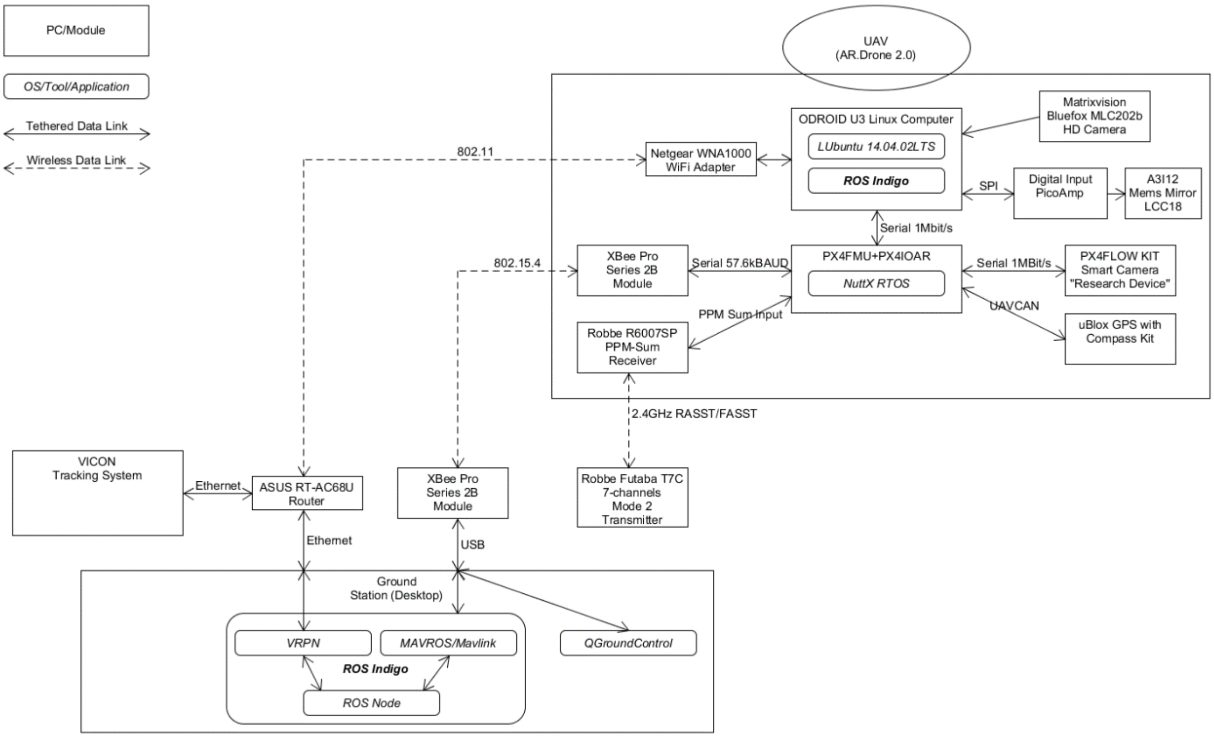
Listing 14: Polygon.msg

```
Coordinate[] vertex
```

Listing 15: Status.msg

```
int8 status_code
string reason
```

UFO – Basic System Overview



MAV Setup

ARDrone 2.0 Frame

The ARDrone 2.0 Frame is used as a weight optimized frame in combination with the PX4 platform. For full PX4 platform setup the PX4FMU and the PX4IOAR (don't interchange with PX4IO) board are required.

Assembly instructions about the PX4IOAR in combination with the ARDrone 2.0 frame can be found at: https://pixhawk.org/platforms/multicopters/ar_drone

The parts list isn't up to date any more so take a look at Chapter 5 "Ordering List".

PX4IOAR

The PX4IOAR board (https://pixhawk.org/modules/px4ioar) is the IO board for making the PX4FMU work on the AR Drone 2.0 frame. Don't mix it up with the common PX4IO board which is a general IO board for small sized aerial vehicles. The AR Drone frame isn't compatible with the PX4IO because internal logics of the IO board can not be adapted easily.

Connection Mapping

In the following an overview of the PX4IOAR's most important connections is shown:

UART 5 (front connector - 4 pins): PX4FLOW

UART 1 (2 front rows connectors - 4 pins needed): Xbee or Lairdtech module (pin slots for easy insertion/removal of the Xbee are not included in the PX4IOAR kit!)

MOTOR TX (side connector - 12pins): motor TX plug from AR Drone 2.0 frame

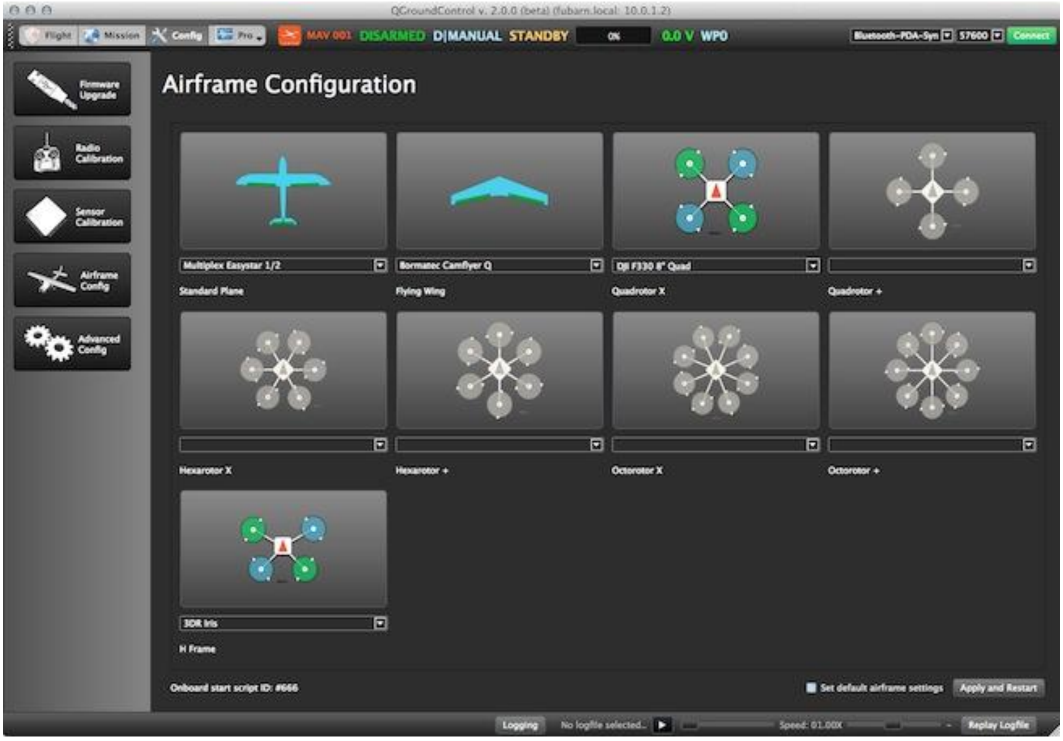
The following figure shows the current drone configuration with PX4IOAR and PX4FMU attached to the AR Drone 2.0 frame.

PX4FMU

The PX4FMU is attached to the PX4IOAR, includes the Flight Controller and communicates with the remote control. Controlling the PX4 Flight platform without appropriate Transmitter is possible but strongly not recommended for safety reasons!

Setup Steps:

- Remove SD Card
- Attach the PX4FMU to the fully assambled AR Drone 2.0 frame + PX4IOAR combination
- Download and setup QGroundcontrol - Setup, Firmware Flash, etc. is recommeded to be done in Windows 7/10 because UART communication seemed to work more flawlessly! Although setting parameters and configuration of the PX4FMU afterwards was also successfully tested in Ubuntu 14.04.02LTS without serious problems:
    - Download QGroundcontrol stable version from: http://qgroundcontrol.org/downloads
    - Download and install the PX4 Driver also listed on this page.
    - Use the USB Cable to connect the PX4 based AR Drone 2.0 (you don’t need power supply for the PX4IOAR and it has not to be switched on!)
    - After drivers are installed successfully install QGRoundcontrol
    - Start QGroundcontrol, check for valid COM port and connect to the PX4FMU
    - First important thing to do is CHOOSE the CORRECT FRAME (AR Drone frame in Menue “Config” → “Airframe Config” → “Quadrotor X” → AR Drone 2.0 Frame



  - Then navigate to Firmware upgrade, disconnect Connection from the PX4FMU and remove the USB cable.
  - Select stable firmware branch and follow the recommended steps to do firmware upgrade.
- After successfull firmware upgrade restart the FMU by disconnecting from QGroundcontrol and removing/plugging in the USB cable. During this step, before again connecting the USB cable, also the SD and battery pack or power supply can be attached. Connect a battery or power supply with the proper voltage and power specifications (11.1V).
- For detailled purpose of the SD card like placing configuration files and evaluating logs please also refer to: https://pixhawk.org/dev/system_startup
- The PX4IOAR board can now be switched on with the ON/OFF switch.
- After the FMU fired up successfully (you should hear a melodic sound followed by by test shaking of all 4 propellers) you want to continue with Sensor Calibration.
- Navigate to “Config” → “Sensor Calibration”
- Execute step by step calibration of the single sensor groups in the following order:
  - Gyroscope Calibration
  - Accalerometer Calibration
  - Compass Calibration
  - DURING ACCALEROMETER CALIBRATION WATCH OUT FOR PLANE SURFACES. The 3D view of the aerial vehicle might not change and tell you the individual steps necessary but all 6 planes have to be covered to achieve successfull calibration. Rest positions are autodetected and every successfull calib. step is confirmed with a short beep sound. After fully completed a short melody is played. You could also check calibration by inspecting the HUD in QGroundcontrol when connected to the PX4FMU.
- If all sensor calibration steps were completed successfully, you may want to continue with TX/RX setup which is given in Chapter 2.

Parameters Tuned via QGroundcontrol:

- INAV:
- INAV_W_Z_BARO = 0

1.4 GPS Module and Compass

The PX4 platform is capable of fusing GPS data for outdoor navigation. Details on the 3DR uBlox LEA-6H module can be found at: https://pixhawk.org/peripherals/sensors/gps

The GPS module should be connected directly to the PX4FMU controller rather than to the PX4IOAR. The proposed interface is UART 6 (ttyS5).

Configuration of the GPS module is done by the following steps:

- Rotate the GPS module so that the arrow “FRONT” points to the front
- Mount the module on top of the Pixhawk FMU
    - Put some spacing between FMU and GPS module (ideally, a GPS mast)
    - I just glued a wooden block on top of the FMU and put the module on top of the block
- Connect the GPS connector of the GPS module with the FMU (use the plug at the left-back of the FMU, the only one that fits)
- Remove the magnetometer cable, you won’t need this
- Now the MAV is ready to fly!
- To enable GPS control during flight (for my RC configuration):
    - Start in maunal mode (switch on the left-top (channel 7) of the RC pushed away)
    - Change switch on left-top to middle position (altitude/position control mode)
    - Pull switch on right top (channel 5) to switch from altitude control to position control

1.5 PX4FLOW

The PX4FLOW Sensor is capable of estimating altitude by ultra sonic sensor and velocity over ground in x -and y direction. The PX4FLOW’s camera should be properly focused before use. Experiments showed that velocity estimations strongly depend on lightning conditions. In outdoor environments on properly textured surfaces the estimates are quite acceptable. Indoors without proper light sources the estimates are unacceptable even if enhanced “low lighting conditions” mode is enabled.

1.6 XBee Communication

The Xbee Module (recommended one is the S2B Pro with 63mW power and wired antenna) is used for sending/receiving low bandwidth data with very low latency. The communication interface basically consists out of one groundstation-attached module (which is attached via USB Explorer Board to the groundstation PC) and one or multiple MAV attached modules. For detailed setup of the modules please refer to: https://pixhawk.org/peripherals/radio-modems/xbee

It is important to mention that in terms of 2 way communication (sending and receiving) the bandwidth should be set to 57600Byte/s max. Otherwise communication may get unreliable. A good alternative with larger bandwidth may be the Lairdtech Mdoules but haven’t been tested yet (also refer to http://www.mouser.at/new/lairdtechnologies/laird-RM024/).

1.7 Companion Computer

The companion computer currently in use is the ODROID XU3 from Hardkernel (http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127) which can be used to communicate to the PX4FMU via its serial port. It is also connected to the Groundstation via WiFi Stick.

The current setup of the Companion computer includes:

- WiFi Nano Stick Netgear WN1000 (working out of the box)
- Optional: DLink DWA171 Wifi Nano Stick (not compatibel with 14.04LTS Kernel yet)
- Up to two BlueFox Cameras rgb/grey-scle MLC202b from MatrixVision
- SanDisk SD-Card
- DC Regulator for DC conversion including Cooling: either the … or the SMO DC-DC Converter http://www.amazon.de/SMO-Konverter-Step-Regulator-Netzteil/dp/B00RWZ2XTW/ref=sr_1_4?ie=UTF8&qid=1426502009&sr=8-4&keywords=smo+dc-dc+konverter
-

Basic setup steps:

- The ODROID XU3 needs 5V power supply with up to 4A current so battery supply voltage of the PX4IOAR has to be stepped down by DC/DC conversion. Because of the quite high power dissipation of the DC/DC converter the cooling policy should be changed after (INSERT LINK TO SCRIPT HERE) startup. The DCDC converter should be placed near the active cooler of the XU3.
- For OS installation, there are preconfigured LUbuntu 14.04LTS images which can be found at: http://www.odroid.in/ubuntu_14.04lts/
- The image has to be written to the SD-card and has to be inserted in the proper slot before startup. Recommended minimum size is 32GByte and transfer rates should be as high as possible. Currently the use of EMMC Modules is evaluated.
- After installing the LUbuntu Image ROS Indigo Full Desktop can be installed on the Odroid XU3.
- Install additional packages as necessary

Setting up camera interface, e.g. for PTAM:

- Install Kumar Bluefox package (driver + important packages already included!)
- Look for camera hardware ID and try interfacing it via image_viewer package, ID has to be given in the launch file
- Use wxPropTool to setup and basically calibrate camera
- Use boost command (boost:=true) to achieve higher framerates
- Calibrate the camera using the PTAM calibration tool
- Start PTAM

Imprtant notes on the USB Ports:

- The Odroid has 2 USB 2.0 Controllers per definition, but 1 is already used for the Ethernet Interface! So all the 4 USB Connectors on the side where also the Ethernet plug is located share one USB 2.0 Controller!
- The Odroid XU3 has 1 USB 3.0 Root Hub (according to the “lsusb --tree” command) however the overall bandwidth for the common USB 3.0 port and the OTG port is limited to 100MByte/s.

- Plug on/off devices to get an idea how they are attached/configured by checking all devices listed in the system: “dmesg”

2. Remote Control/Receiver Setup

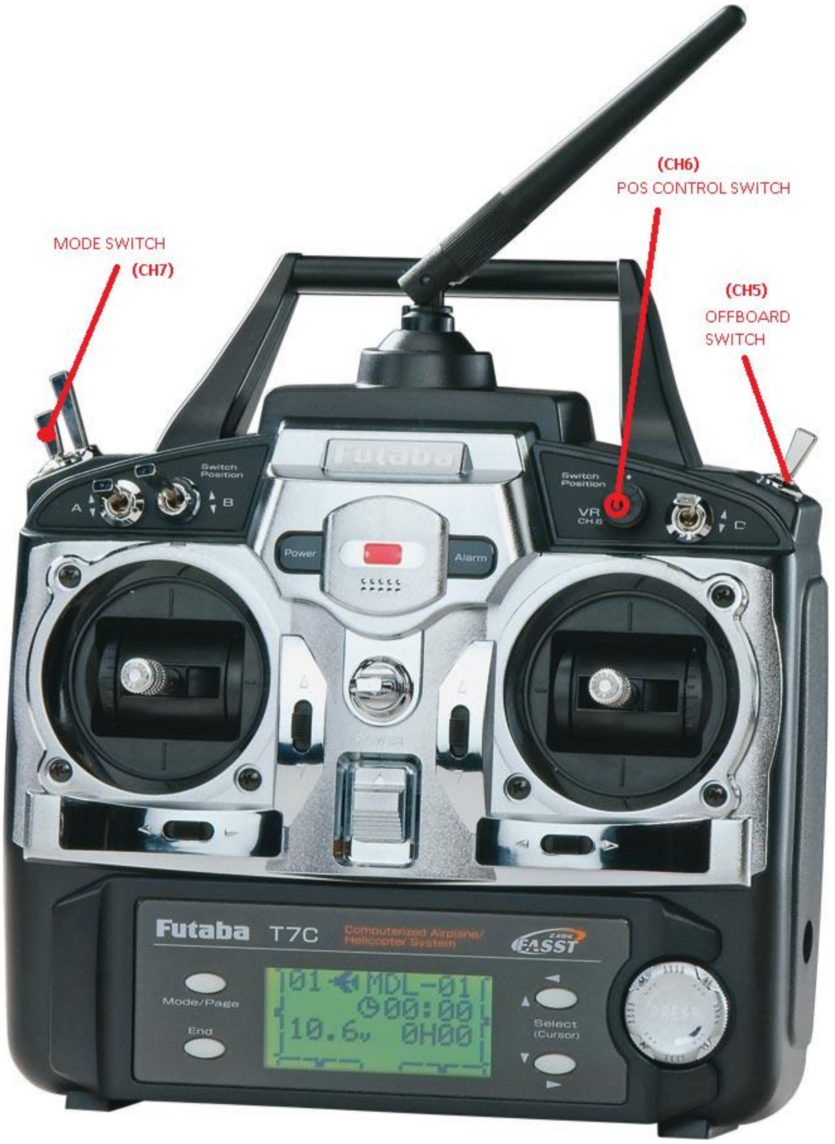
The Remote Control and Receiver setup recommended is the Robbe Futaba T7C Tx and the R6007SP Rx. The R6007SP directly outputs a PPM sum signal and linking to the RC, connection to the FMU and calibration/control work flawlessly. Besides the R6007SP weighs 3g only.

2.1 Linking the R6007SP to the T7C Transmitter

- Attach the R6007SP to the PX4IOAR (standard Robbe Futabe connectors can be use, watch out for proper pin connection)
- Turn on T7C Transmitter
- Place it near the R6007SP
- Hold the easy link button BEFORE switching on power supply at the PX4IOAR
- Switch on power
- Wait 2-3 sec until the R6007SP's LED stops flashing red but turns green
- If the LED turns green the R6007SP Receiver is properly linked to the T7C Transmitter

2.2 Radio Calibration

- Switch on the PX4IOAR and wait until FMU is ready
- Switch on remote control and wait until the receiver's LED turns green
- Connect the USB calbe to the PC with QGroundcontrol
- Start QGroundcontrol and go To "Config" → "Radio Calibration"
- Follow the steps and use recommended mapping of switches:
  - MODE SWITCH → CH7
  - POSITION CONTROL SWITCH → CH6
  - OFFBOARD SWITCH → CH5



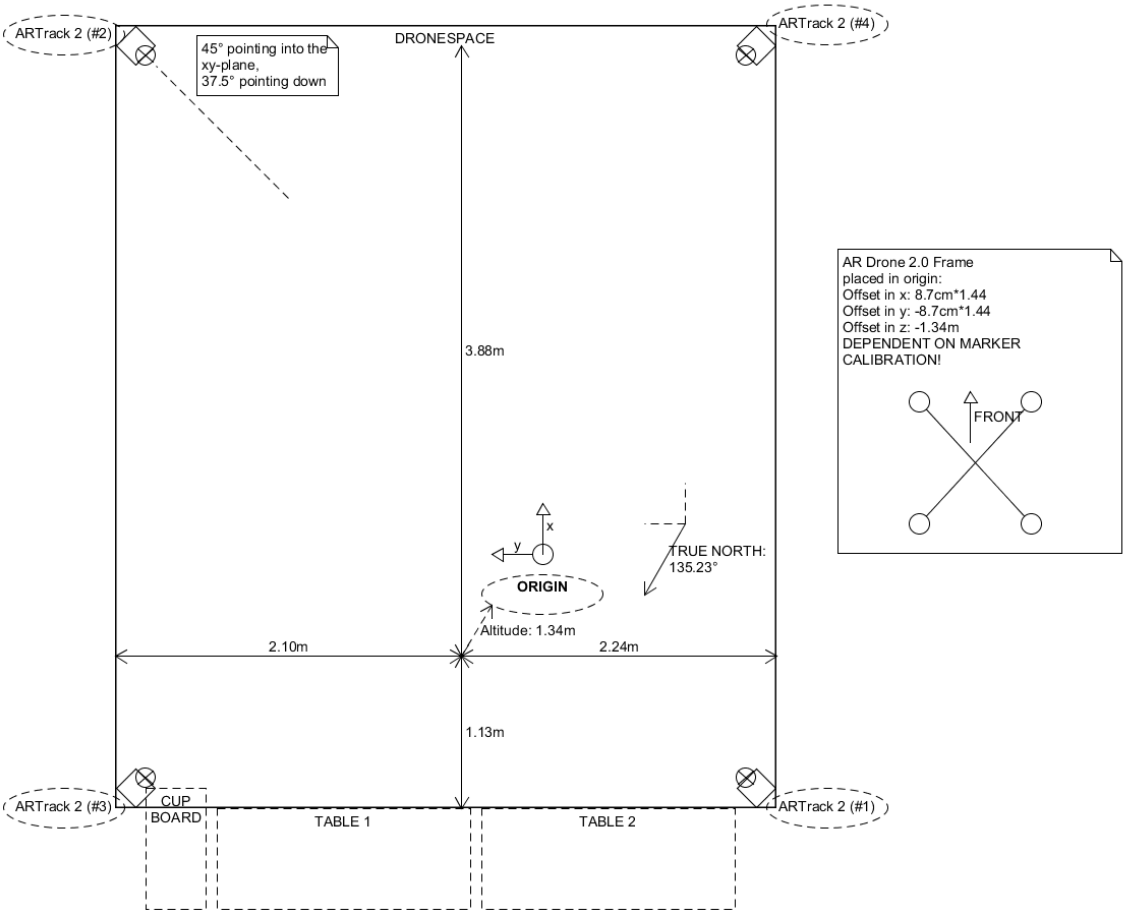
It is mentionable that also if velocity control mode is used in OFFBOARD mode, then the POS CONTROL SWITCH should be left in ENABLED position. To enable offboard mode in general no other control mode has to be necessarily enabled! Streaming valid waypoints has to be done at a recommended rate of at least 2Hz (4Hz preferable).

3. ART Tracking System & Flying Arena

The ART Tracking system is capable of tracking one or multiple objects in 3D space using reflective markers. Per object orientation and position (6DOF) can be measured with sub-mm accuracy depending on the Cameras Lenses and calibration. The maximum tracking rate is up to 60Hz.

3.1 Current Setup

The Dronespace provides a theoretical flying space of about 4x5x3m with an effective tracking space of about 3x4x2.25m. Currently we use 4 ARTrack 2 Cameras which are mounted in each Corner of the Dronespace (hanging position). With 37.5° angle looking down from above and 45° turned into the Dronespaces plane area. The ARTrack 2 cameras are equipped with wide FOV lenses which means (88°x58° HxV FOV and 4.5m distance → also refer to http://www.ar-tracking.com/products/discontinued/arttrack2/). The remaining ARTrack 3 Cameras which are not used at the moment have a small FOV with 52°x35° FOV and 7m distance.



3.2 DTrack 1 Software & VRPN Server Setup

The DTrack 1 application (DTrack.exe can be found at the Desktop) is necessary to communicate with the ART Tracking cameras in realtime over Ethernet. Therefore a PCI synch card is necessary which must be connected in line with the cameras via BNC cables, including a termination resistor at the end. So each camera has a power connector (watch out for correct power supplies and don’t mix with ARTrack 3 supplies!), a synch cable connector (BNC) and a ethernet patch cable connector for data transfer. The DTRack 1 is used for the following setup steps which also should be executed in the following order:

- Setting up the camera parameters properly (amount, orientation, Tracking rate, Intensity, ...)
- Aiming the cameras: Use Monitor tool to center the view of cameras as necessary (green, yellow and red markers show quality of tracking!)
- Calibration of the Room (Raindance): Two markers are needed, the coordinate system and the 420mm T marker
- Calibration of object(s): Place the object at zero position and do raindance with the object (no coordinate system maker is needed)
- It is mentionable that at least 2 cameras have to “see” the object to not loose tracking. If tracking is lost once, it can take quite a long time to recover!). This is also valid for calibration. Always put the coordinate marker so that at least 2 cameras can see it.

The VRPN Server is a middleware which communicates with the DTrack software on one hand (the DTrack software just broadcasts tracking data in string format into the 172.28.x.x network!) and any VRPN client based application on the other hand. Mutliple trackers/tracking data can be extracted via VRPN. For detailed information about VRPN refer to:

http://www.cs.unc.edu/Research/vrpn/

or geeks refer to:

http://www.vrgeeks.org/vrpn

Current version which is used and already properly setup and working is the 7.30 version. It is also strongly recommended on client side so that no compatibility issues occur.

3.3 Network Setup

Basically the setup can be divided into two main networks:

- The ART Tracking network plus the Groundstation/MAV network which is located in the 172.28.x.x network (subnet address is 255.255.0.0).
- The network to provide internal access to the groundstation or guests in the Dronespace. It is located in the 192.168.0.x network (subnet address is

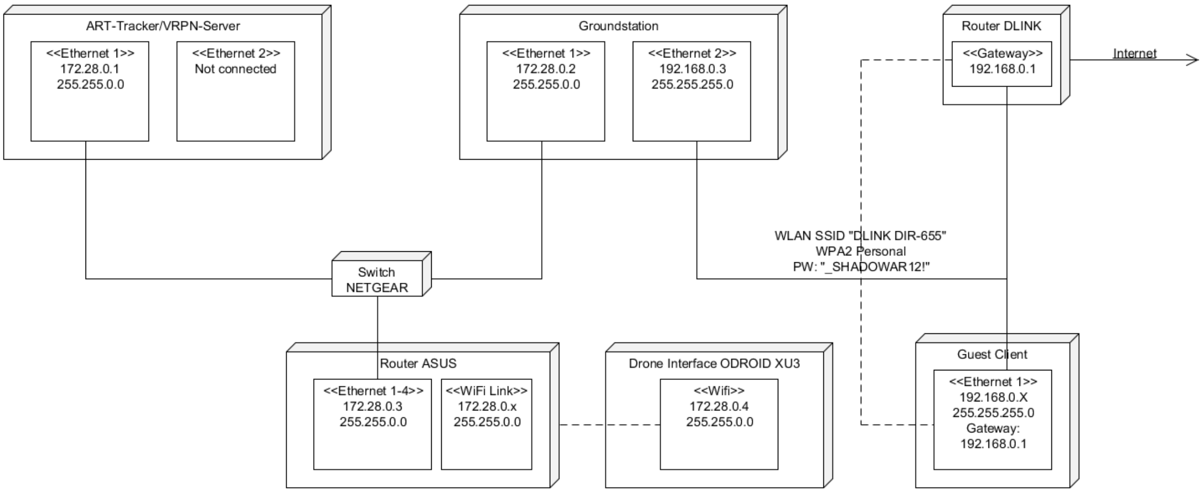
255.255.255.0). To access via WiFi connect to the DLINK DIR655 network, setup IP config properly. Password is thereby _SHADOWAR12!

ARTTracking network:

- DTRack 1/VRPN Server: 172.28.0.1 - 255.255.0.0
- Groundsation Server: 172.28.0.2 - 255.255.0.0
- ASUS AC68RT Wireless Router: 172.28.0.3 - 255.255.0.0
- Handheld client for VRVU: 172.28.0.10 - 255.255.0.0

Network for internet access:

- Router is the DLink DIR655: 192.168.0.1 - 255.255.255.0
- Groundsation: 192.168.0.2 - 255.255.255.0
- UFO Project Laptop: 192.168.0.3 - 255.255.255.0


ART-Tracker/VRPN-Server
<<Ethernet 1>> 172.28.0.1 255.255.0.0
<<Ethernet 2>> Not connected
Groundstation
<<Ethernet 1>> 172.28.0.2 255.255.0.0
<<Ethernet 2>> 192.168.0.3 255.255.255.0
Router DLINK
<<Gateway>> 192.168.0.1
Internet
WLAN SSID "DLINK DIR-655"
WPA2 Personal
PW: "_SHADOWAR12!"
Switch NETGEAR
Router ASUS
<<Ethernet 1-4>> 172.28.0.3 255.255.0.0
<<WiFi Link>> 172.28.0.x 255.255.0.0
Drone Interface ODROID XU3
<<Wifi>> 172.28.0.4 255.255.0.0
Guest Client
<<Ethernet 1>> 192.168.0.X 255.255.255.0 Gateway: 192.168.0.1

4. Groundstation Setup

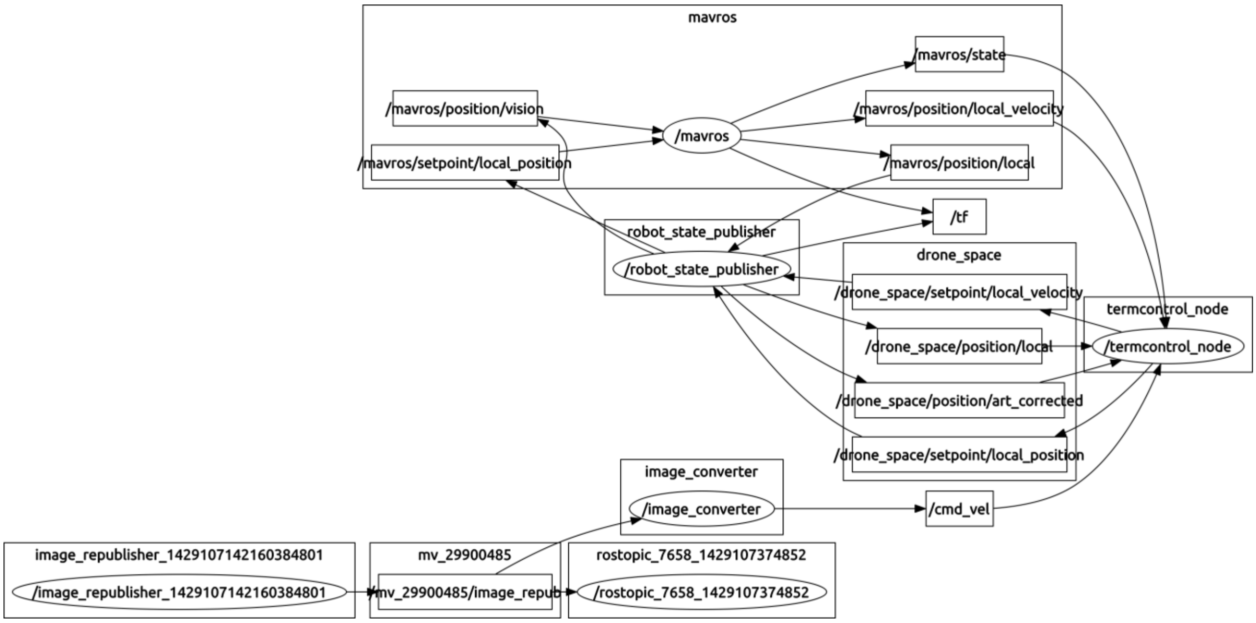
Basically the Groundstation is setup with Ubuntu 14.04LTS (64Bit) whereby the following components were installed:

- ROS Indigo Full Desktop
- QT Creator
- OpenCV 2.4.10
- VRPN 7.30
- NTPD Service

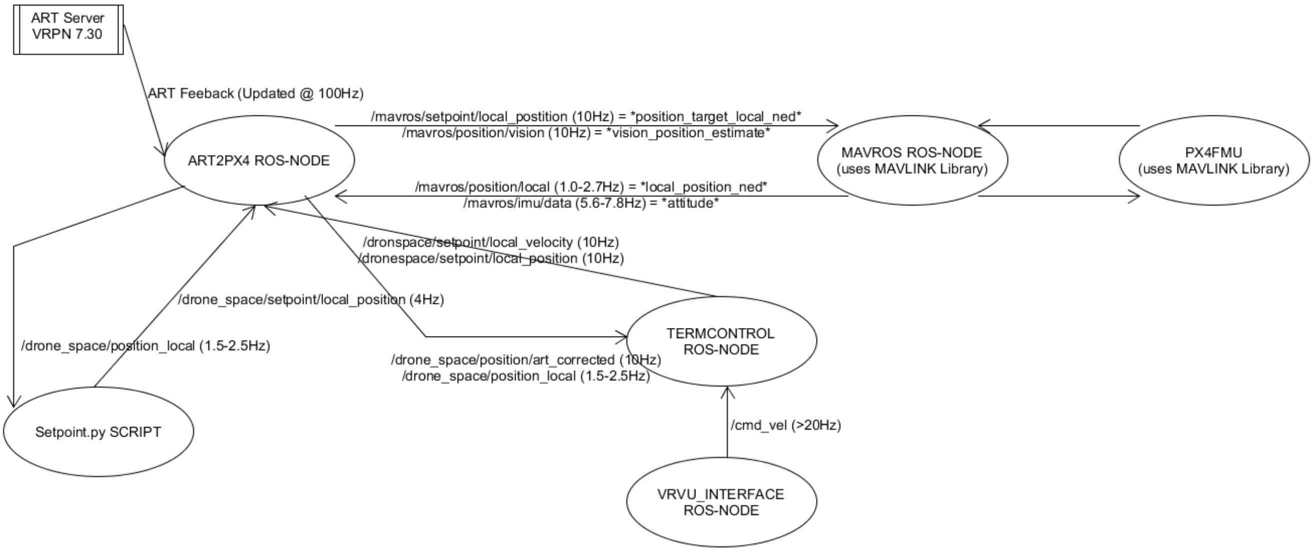
Based on the ROS the following architecture is used to control the PX4 based drone offboards:

4.1 ROS Setup

The nodes of the pixhawk control setup are configured as shown in the following figure:


mavros
/mavros/state
/mavros/position/vision
/mavros
/mavros/position/local_velocity
/mavros/setpoint/local_position
/mavros/position/local
/tf
robot_state_publisher
/robot_state_publisher
drone_space
/drone_space/setpoint/local_velocity
termcontrol_node
/drone_space/position/local
/termcontrol_node
/drone_space/position/art_corrected
/drone_space/setpoint/local_position
image_converter
/image_converter
/cmd_vel
image_republisher_1429107142160384801
/image_republisher_1429107142160384801
mv_29900485
/mv_29900485/image_repub
rostopic_7658_1429107374852
/rostopic_7658_1429107374852

Considering the rates of the topics, the following image gives an overview:

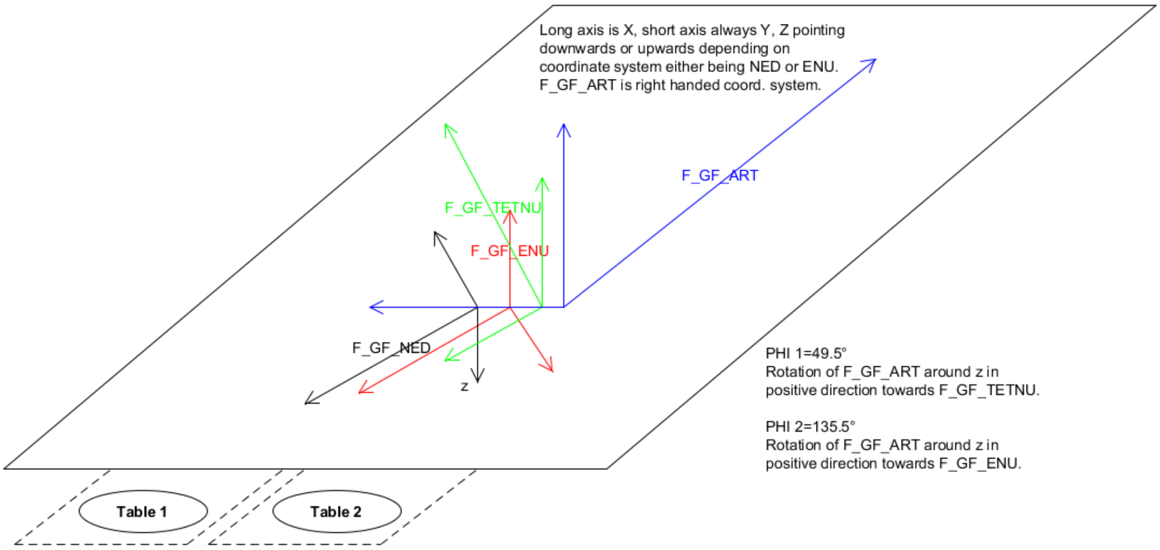

ART Server
VRPN 7.30
ART Feeback (Updated @ 100Hz)
ART2PX4 ROS-NODE
/mavros/setpoint/local_postition (10Hz) = *position_target_local_ned*
/mavros/position/vision (10Hz) = *vision_position_estimate*
MAVROS ROS-NODE
(uses MAVLINK Library)
PX4FMU
(uses MAVLINK Library)
/mavros/position/local (1.0-2.7Hz) = *local_position_ned*
/mavros/imu/data (5.6-7.8Hz) = *attitude*
/dronspace/setpoint/local_velocity (10Hz)
/dronespace/setpoint/local_position (10Hz)
/drone_space/setpoint/local_position (4Hz)
/drone_space/position_local (1.5-2.5Hz)
TERMCONTROL
ROS-NODE
/drone_space/position/art_corrected (10Hz)
/drone_space/position_local (1.5-2.5Hz)
/cmd_vel (>20Hz)
Setpoint.py SCRIPT
VRVU_INTERFACE
ROS-NODE

Basically the ART Tracking Software and the VRPN Server run on a Windows XP SP3 machine which is attached to the 172.28.x.x network. The tracking info is also broadcasted into the 172.28.x.x network so any client which has to access tracking data via VRPN can connect either via Wireless network of the AC68RT or directly connect to the Router via network cable.

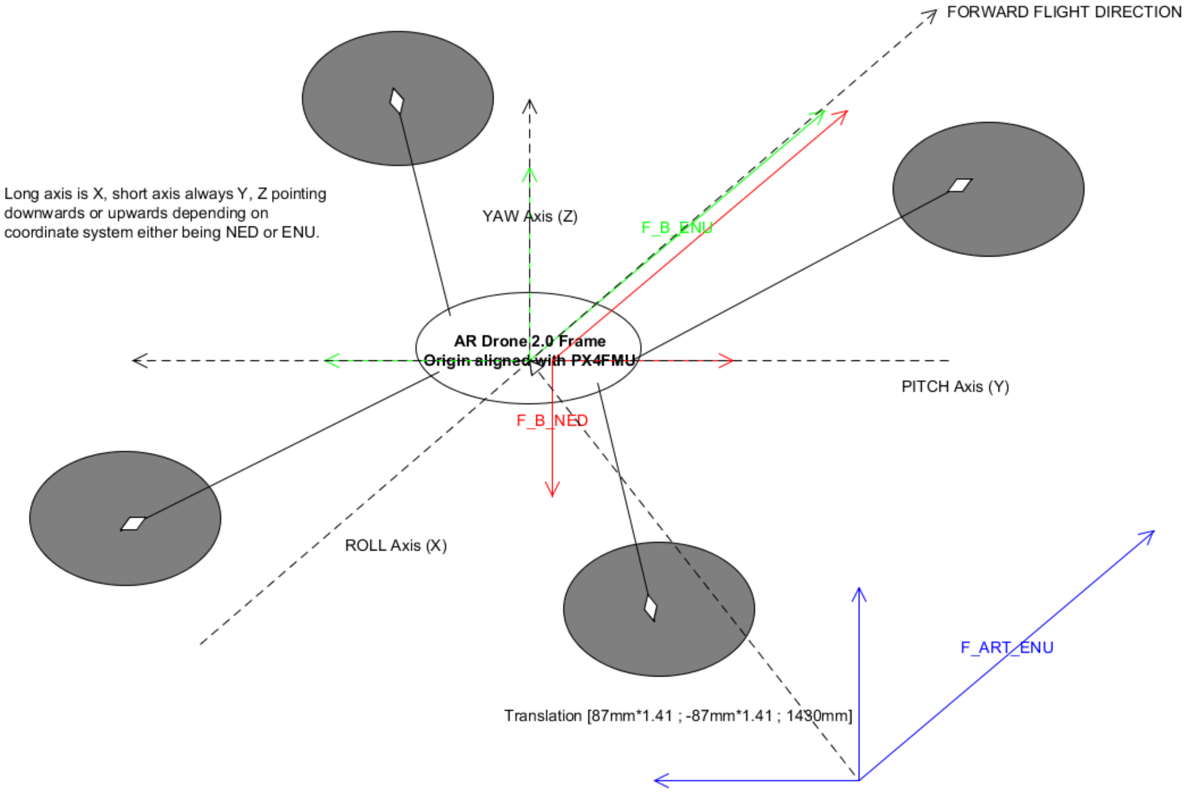
4.4 Coordinate Frames

Orientation overview of droneSpace:

4.4.1 Groundframes


Long axis is X, short axis always Y, Z pointing downwards or upwards depending on coordinate system either being NED or ENU.
F_GF_ART is right handed coord. system.
F_GF_ART
F_GF_TETNU
F_GF_ENU
F_GF_NED
z
PHI 1=49.5°
Rotation of F_GF_ART around z in positive direction towards F_GF_TETNU.
PHI 2=135.5°
Rotation of F_GF_ART around z in positive direction towards F_GF_ENU.
Table 1
Table 2

4.4.2 Bodyframes


FORWARD FLIGHT DIRECTION
Long axis is X, short axis always Y, Z pointing downwards or upwards depending on coordinate system either being NED or ENU.
YAW Axis (Z)
F_B_ENU
AR Drone 2.0 Frame
Origin aligned with PX4FMU
PITCH Axis (Y)
F_B_NED
ROLL Axis (X)
F_ART_ENU
Translation [87mm*1.41 ; -87mm*1.41 ; 1430mm]

4.4.3 Dronespace ROS Messages

Message containing IMU data which is received from the PX4FMU:

/mavros/imu/data → orientation of F_B_ENU with respect to F_GF_ENU

Message which is containing actual orientation/position of the MAV measured by the PX4FMU:

/mavros/position/local → orientation of F_B_ENU with respect to F_GF_ENU

translation of F_B_ENU with respect to F_GF_TETNU

Message to forward the feedback data (ART Tracking, Vision SLAM etc.):

/mavros/position/vision → orientation of F_B_ENU with respect to F_GF_ENU

translation of F_B_ENU with respect to F_GF_TETNU

Message to send position target commands:

/mavros/setpoint/local_position → orientation of F_B_NED with respect to F_GF_NED

translation of F_B_ENU with respect to F_B_TETNU

Message to send position target commands but already "corrected" to the ART frame:

/dronespace/setpoint/local_position → orientation of F_B_ENU with respect to F_ART

translation of F_B_ENU with respect to F_ART

Messages to receive actual position of MAV but already “corrected” to the ART frame:

/dronespace/position/local → orientation of F_B_ENU with respect to F_ART
translation of F_B_ENU with respect to F_ART

/dronespace/position/art_corrected → orientation of F_B_ENU with respect to F_ART
translation of F_B_ENU with respect to F_ART

4.5 Control Modes

4.5.1 Manual mode (Arming)

In manual mode the PX4FMU takes raw commands from the Tx (T7C) only. Although basic stabilization of Roll and pitch are achieved, to hold the MAV parallel to the ground.

4.5.2 Altitude control

In altitude control mode the PX4FMU either considers the estimates of the barometric pressure sensor, or if available estimates from the vision system to stabilize vertical position. Via Throttle commands of the TX it is possible to control the altitude and the MAV should stabilize itself in the targeted height. It is noticeable that in indoor environments the barometric pressure sensor is not suitable for altitude control because of serious drift over time. When testing Altitude control, by sending position tracking estimates to the FMU, the performance was quite fine.

5. Ordering List

For a detailed ordering list please refer to:

https://docs.google.com/spreadsheets/d/1JgFE7KcxnEcFsMJFW3amYkZtSzkSPRpP5rB3HOCRrks/edit?pli=1

6. Weight Table

For a weight overview of the components used please refer to:

https://docs.google.com/spreadsheets/d/1OHTCkvnHNJgLWdz4S9PWDYAkK6c-yXpbCcRbS0R5lhk/edit?usp=sharing

ARDUINO IMPLEMENTATION OF THE LASER DRIVER

```
#include <Tone.h>

//Laser through serial

#include <ros.h>
#include <stdio.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float32MultiArray.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>
#include <SPI.h>

// ROS stuff
ros::NodeHandle nh;
std_msgs::String str_msg;
std_msgs::Float32 ack_msg;
ros::Publisher chatter("acknowledgment", &ack_msg);
//ros::Publisher chatter_verbose("acknowledgment_verbose", &str_msg);

//Tone STUFF
Tone tone1;

//LASER stuff
// set pin 10 as the slave select for the digital pot:
const int slaveSelectPin = 10;
const int sixtykHzPin = 4;
const int HVenablePin = 2;
unsigned int DAC_ZERO_LEVEL=32767;
```

```
float MIRROR_INC_MAX_DEG=4.5;

int datasize = 0;


void messageCb( const std_msgs::Float32MultiArray& msg){


 datasize = msg.data[0];

 ack_msg.data = datasize;

 chatter.publish( &ack_msg );




 if(msg.data[1] == 10 && msg.data[2] == 10){ //since 0 position is the size


  WritePicoAmpXY(0,0); // put HVpin down

  delay(10);




  digitalWrite(HVenablePin,LOW);




 } else

  {




 // LASER stuff




 // ENABLE HIGH VOLTAGE!



 digitalWrite(HVenablePin,HIGH);
```

```
delay(1);

// DRAW VECTOR STUFF


for(int i=1;i<datasize;i = i + 2)


{

 // str_msg.data = "Start drawing";

 // chatter_verbose.publish( &str_msg );




  /*WritePicoAmpXY(msg.data[0],msg.data[1]);

  delay(4);

  WritePicoAmpXY(msg.data[2],msg.data[3]);

  delay(4);

  WritePicoAmpXY(msg.data[4],msg.data[5]);

  delay(4);

  WritePicoAmpXY(msg.data[6],msg.data[7]);

  delay(4);*/


  //str_msg.data = "Finish draw";

  //chatter_verbose.publish( &str_msg );


  WritePicoAmpXY(msg.data[i],msg.data[i+1]);

  ack_msg.data = msg.data[i]*100 + msg.data[i+1];//i;

  chatter.publish( &ack_msg );


  delay(4);
  }


    }


}


ros::Subscriber<std_msgs::Float32MultiArray> sub("coordinates", &messageCb );



void setup()
{
  //pinMode(9, OUTPUT);

  nh.initNode();
  nh.subscribe(sub);
  nh.advertise(chatter);
  //nh.advertise(chatter_verbose);

  digitalWrite(HVenablePin,LOW);
  pinMode (HVenablePin, OUTPUT);
  digitalWrite(HVenablePin,LOW);

  // INITIALIZE SPI:
  //pinMode (slaveSelectPin, OUTPUT); // set the slaveSelectPin as an output:
  SPI.setBitOrder(MSBFIRST); // or MSBFIRST according to the AD5664R
  SPI.setDataMode(SPI_MODE1); // SPI_MODE0 up to SPI_MODE3 see also
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus clocl polarity and phase
  //SPI.setClockDivider(SPI_CLOCK_DIV64); // Uno runs at 16MHz which gives a SPI  clock of 1MHz
  SPI.setClockDivider(SPI_CLOCK_DIV16); // Uno runs at 16MHz which gives a SPI  clock of 1MHz
  SPI.begin();
  delay(100);


//  Recommended register downloads for Pico Amp:
//  2621441 Dec = 280001 Hex FULL RESET
//  3670017 Dec = 380001 Hex ENABLE INTERNAL REFERENCE
//  2097167 Dec = 20000F Hex ENABLE ALL DAC CHANNELS
//  3145728 Dec = 300000 Hex ENABLE SOFTWARE LDAC
  digitalAmpWrite(0x28,0x00,0x01);
  delay(100);


  digitalAmpWrite(0x38,0x00,0x01);
  delay(100);


//  digitalAmpWrite(0x20,0x00,0x08); //enable all dac channels
//  delay(100);
//  digitalAmpWrite(0x20,0x00,0x07);
//  delay(100);
  digitalAmpWrite(0x20,0x00,0x0F); //enable all dac channels
  delay(100);



  // Set all channels to 1.25V bias (direct write to DAC channel register)
  digitalAmpWrite(0x1F,0x7F,0xFF);
  delay(10000);


  //digitalAmpWrite(0x30,0x00,0x0F); // Set LDAC mode 1111
  // After DAC is enabled, send 32768 digital input to all four channels:
  // 32768 DEZ = 0x8000
  //Serial.begin(9600);


  tone1.begin(8);
  tone1.play(60000);
}


void loop()
{
  //ROS stuff
  nh.spinOnce();
  delay(1);


}


void WritePicoAmpXY(float x_deg, float y_deg)

{


  unsigned int DAC_OUT_VALUE_X_PLUS=  DAC_ZERO_LEVEL;


  unsigned int DAC_OUT_VALUE_X_MINUS= DAC_ZERO_LEVEL;


  unsigned int DAC_OUT_VALUE_Y_PLUS=  DAC_ZERO_LEVEL;


  unsigned int DAC_OUT_VALUE_Y_MINUS= DAC_ZERO_LEVEL;





  if(fabs(x_deg)/MIRROR_INC_MAX_DEG <= 1.0)
```

```
 {

   DAC_OUT_VALUE_X_PLUS  =  DAC_ZERO_LEVEL + int(x_deg/MIRROR_INC_MAX_DEG*32767.0);

   DAC_OUT_VALUE_X_MINUS =  DAC_ZERO_LEVEL - int(x_deg/MIRROR_INC_MAX_DEG*32767.0);



   if(DAC_OUT_VALUE_X_PLUS < 65535 && DAC_OUT_VALUE_X_PLUS > 0 && DAC_OUT_VALUE_X_MINUS < 65535 && DAC_OUT_VALUE_X_MINUS > 0)

   {

digitalAmpWrite(0x18,Int16ToHighByte(DAC_OUT_VALUE_X_PLUS),Int16ToLowByte(DAC_OUT_VALUE_X_PLUS));

digitalAmpWrite(0x19,Int16ToHighByte(DAC_OUT_VALUE_X_MINUS),Int16ToLowByte(DAC_OUT_VALUE_X_MINUS));

   }

 }



 if(fabs(y_deg)/MIRROR_INC_MAX_DEG <= 1.0)

 {
   DAC_OUT_VALUE_Y_PLUS  =  DAC_ZERO_LEVEL + int(y_deg/MIRROR_INC_MAX_DEG*32767.0);

   DAC_OUT_VALUE_Y_MINUS =  DAC_ZERO_LEVEL - int(y_deg/MIRROR_INC_MAX_DEG*32767.0);



   if(DAC_OUT_VALUE_Y_PLUS < 65535 && DAC_OUT_VALUE_Y_PLUS > 0 && DAC_OUT_VALUE_Y_MINUS < 65535 && DAC_OUT_VALUE_Y_MINUS > 0)

   {

digitalAmpWrite(0x1A,Int16ToHighByte(DAC_OUT_VALUE_Y_PLUS),Int16ToLowByte(DAC_OUT_VALUE_Y_PLUS));

digitalAmpWrite(0x1B,Int16ToHighByte(DAC_OUT_VALUE_Y_MINUS),Int16ToLowByte(DAC_OUT_VALUE_Y_MINUS));

   }

  }



}

byte Int16ToHighByte(unsigned int Integer16high)

{
  unsigned int tmp_int=Integer16high >> 8;

  byte tmp_highbyte=(byte)tmp_int;

  return tmp_highbyte;

}

byte Int16ToLowByte(unsigned int Integer16low)

{

  byte tmp_lowbyte=byte(Integer16low);

  return tmp_lowbyte;

}



// SPI Write 24-Bit Command interface

//===================================

void digitalAmpWrite(char byte2, char byte1, char byte0)

{
  // take the SS pin low to select the chip:

  digitalWrite(slaveSelectPin,LOW);

  //  send in the address and value via SPI:

  SPI.transfer(byte2);

  SPI.transfer(byte1);

  SPI.transfer(byte0); // Change order to fulfill MSB first convention

  // take the SS pin high to de-select the chip:

  digitalWrite(slaveSelectPin,HIGH);

}
```

REFERENCES

1. Mell, Peter, and Timothy Grance. "The NIST definition of cloud computing (draft)." NIST special publication 800.145 (2011): 7.

2. Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." *Computer networks* 54.15 (2010): 2787-2805.

3. Sanfeliu, Alberto, Norihiro Hagita, and Alessandro Saffiotti. "Network robot systems." Robotics and Autonomous Systems 56.10 (2008): 793-797.

4. Chibani, A., et al. "Ubiquitous robotics: Recent challenges and future trends." Robotics and Autonomous Systems (2013).

5. Chen, Yinong, Zhihui Du, and Marcos García-Acosta. "Robot as a service in cloud computing." Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on. IEEE, 2010.

6. Quintas, J., P. Menezes, and J. Dias. "Cloud robotics: towards context aware robotic networks." *International Conference on Robotics*. 2011.

7. Kamei, Koji, et al. "Cloud networked robotics." Network, IEEE 26.3 (2012): 28-34.

8. Waibel, Markus, et al. "Roboearth." *Robotics & Automation Magazine, IEEE*18.2 (2011): 69-82.

9. Nonami, Kenzo, et al. Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles. Springer Publishing Company, Incorporated, 2010.

10. Austin, Reg. Unmanned aircraft systems: UAVS design, development and deployment. Vol. 54. Wiley. Com, 2011.

11. (Brecher, December 2003.)

12. Peschel, Joshua Michael, and Robin Roberson Murphy. "On the Human–Machine Interaction of Unmanned Aerial System Mission Specialists." Human-Machine Systems, IEEE Transactions on 43.1 (2013): 53-62.

13. Williams, Kevin W. An assessment of pilot control interfaces for unmanned aircraft. No. DOT/FAA/AM-07/8. FEDERAL AVIATION ADMINISTRATION OKLAHOMA CITY OK CIVIL AEROMEDICAL INST, 2007.

14. Murphy, R., and J. Burke. "From remote tool to shared roles." Robotics & Automation Magazine, IEEE 15.4 (2008): 39-49.

15. ARINC Aereonautical Radio. http://www.aviation-ia.com/aeec/projects/cds/index.html

16. Cummings et al. , J.T.: STANG 4586 Human Supervisory Control Implications. In Proceedings of UVS Canada Annual Conference, Montebello, Quebec, Canada (2006)

17. NATO: The North Atlantic Treaty Organization(2007),http://www.nato.int/structur/AC/224/standard/AEDP2/AEDP2_Documents/AEDP-02v1.pdf

18. RTCA: Radio Technical Commission for Aeronautics, and the European Organization for Civil Aviation Equipment EUROCAE (1992), http://www.do178site.com/

19. Society of Automotive Engineers SAE, Unmanned Systems Technical Committee AS4 (2010), http://standards.sae.org/as5684a/

20.Society of Automotive Engineers SAE, Unmanned Systems Technical Committee AS4 (2010), http://standards.sae.org/as5684a/

21.ISO: International Organization for Standardization (2000), http://www.iso.org/iso/iso_catalogue/ catalogue_tc/catalogue_detail.htm?csnumber=30030

22.Ponsa, P., Díaz, M.: Creation of an Ergonomic Guideline for Supervisory Control Interface Design. In: Harris, D. (ed.) HCII 2007 and EPCE 2007. LNCS (LNAI), vol. 4562, pp. 137–146. Springer, Heidelberg (2007)

23. N. J. Cooke, H. K. Pederson, O. Connor, J. C. Gorman, and D. Andrews, "Acquiring team-level command and control skill for UAV operation," in Human Factors of Remotely Operated Vehicles, vol. 7, N. J. Cooke, H. L. Pringle, H. K. Pedersen, and O. Connor, Eds. Amsterdam, The Netherlands: Elsevier, 2006, pp. 285–297.

24.R. R. Murphy, K. S. Pratt, and J. L. Burke, "Crew roles and operational protocols for rotary-wing micro-UAVs in close urban environments," in Proc 3rd ACM/IEEE Int. Conf. Human-Robot Interaction, Mar. 2008, pp. 73–80.

25.M. Nas, "The changing face of the interface: An overview of UAS control issues and controller certification," Unmanned Aircraft Technol. Appl. Res. Work. Group 27, 2008. (in Tab. Cit. 19)

26.J. Cooper and M. A. Goodrich, "Towards combining UAV and sensor operator roles in UAV-enabled visual search," in Proc. 3rd ACM/IEEE Int. Conf. Human-Robot Interaction, 2008, pp. 351–358.

27.M. L. Cummings and P. J. Mitchell, "Managing multiple UAVs through a timeline display," in Proc. Amer. Inst. Aeronaut. Astronaut. Infotech@Aerospace Conf., Sep. 2005.

28.Parrot. (2011). Parrot AR.Drone User Guide [Online]. Available: http://www.parrot.com/fr/support/guidesutilisateur/ar.drone_user-guide_ uk.pdf

29. R. Hopcroft, E. Burchat, and J. Vince, "Unmanned aerial vehicles for maritime patrol: Human factors issues," Aus. Def. Sci. Technol. Org., Canberra, Australia, Rep. DSTO-GD-0463, 2006. (in Tab. Cit. 35)

30.J. Cooper and M. A. Goodrich, "Integrating critical interface elements for intuitive single-display aviation control of UAVs," in Proc. SPIE Defense Security Symp., 2006, vol. 6226. (in Tab. Cit. 40)

31.T. Oron-Gilad and Y. Minkov, "Remotely operated vehicles (ROVs) from the bottom-up operational perspective," in Proc. Human-Robot Int. Future Military Oper., 2010, pp. 211–227. (in Tab. Cit. 41)

32.J. L. Weeks, "Unmanned aerial vehicle operator qualifications," U.S. Air Force Res. Lab., Dayton, OH, Rep. AFRL-HE-AZ-TR-2000–2002, 2000. (in Tab. Cit. 42)

33.Skybotix Technologies. (2011). CoaX Coaxial Helicopter Product Inf. Sheet [Online]. Available: http://www.skybotix.com (in Tab. Cit. 43)

34.AirRobot. (2010). AirRobot AR100B Product Inf. Sheet [Online]. Available: http://www.airrobot-uk.com/air-robot-products.htm (in Tab. Cit. 44)

35.Draganfly Innovations, Inc. (2011). Draganflyer X Series Product Inf [On- line]. Available: http://www.draganfly.com/industrial/products.php (in Tab. Cit. 45)

36.AeroVironment, Inc. (2011).UAS: Raven[Online].Available: http://www.avinc.com/uas/small_uas/raven (in Tab. Cit. 47)

37.AAI Textron Systems, Inc. (2011). Shadow Tactical Unmanned Aircraft Syst. [Online]. Available: http://www.aaicorp.com/products/ uas/shadow_family.html (in Tab. Cit. 48)

38.Northrop Grumman Corporation (2011). Fire Scout Product Inf. Sheet [Online]. Available: url-http://www.as.northropgrumman.com/ products/p6mq8bfirescout/index.html (in Tab. Cit. 49)

39.General Atomics Aeronautical Systems Inc. (2011). Remote Video Terminal Product Inf. Sheet [Online]. Available: http://www.gaasi.com/products/ground_control/rvt.php (in Tab. Cit. 52)

40.Turkish Aerospace Industries, Inc. (2011). Anka Product Inf. Sheet [On-line]. Available: http://www.tai.com.tr/taimain.aspx (in Tab. Cit. 54)

41.Israel Aerospace Industries Ltd. (2011). Heron 1 Product Inf. Sheet [On- line]. Available: http://www.iai.co.il/18900-16382-en/BusinessAreas_ UnmannedAirSystems_HeronFamily.aspx?btl=1 (in Tab. Cit 56)

42.Northrop Grumman Corporation. (2011). Global Hawk Product Inf. Sheet [Online]. Available: http://www.as.northropgrumman.com/ products/ghrq4a/index.html (in Tab. Cit. 57)

43.Guo, Cheng, and Ehud Sharlin. "Exploring the use of tangible user interfaces for human-robot interaction: a comparative study." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2008.

44.Monajjemi, Valiallah Mani, et al. "Hri in the sky: Creating and commanding teams of uavs with a vision-mediated gestural interface." Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS’13). 2013.

45.Liew, Chun Fui, and Takehisa Yairi. "Quadrotor or blimp? Noise and appearance considerations in designing social aerial robot." Human-Robot Interaction (HRI), 2013 8th ACM/IEEE International Conference on. IEEE, 2013.

46.M. Burri et al. “Design and Control of a Spherical Omnidirectional Blimp”, Proc. Of the IEEE int. Conf. On Intelligent Robots and Systems

47.Müller, Jörg. Autonomous navigation for miniature indoor airships. Diss. Universitätsbibliothek Freiburg, 2013.

48.Ermacora, Toma, Bona, Chiaberge, Silvagni, Gaspardone, Antonini "A cloud robotics architecture for an emergency management and monitoring service in a smart city environment."

49.Goldberg, Ken, and Ben Kehoe. "Cloud robotics and automation: A survey of related work." EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5 (2013).

50.F. Michahelles, Rob van Kranenburg and Markus Waibel :Enlisting Robots - Once robots are integrated into the Internet of Things, they can perform tasks automatically., Inside the labs column in,RFID Journal, August, 2012

51.Pagallo, Ugo. "Robots in the cloud with privacy: A new threat to data protection?." Computer Law & Security Review 29.5 (2013): 501-508.

52.Kanda, Takayuki, et al. "Abstracting people's trajectories for social robots to proactively approach customers." Robotics, IEEE Transactions on 25.6 (2009): 1382-1396.

53. "Rosbridge: Ros for non-ros users." Proceedings of the 15th International Symposium on Robotics Research. 2011.

54.Open Knowledge Foundation Blog. Defining Open Data. http://tinyurl.com/qh722vn

55.Gutierrez, P., Barrientos, A., del Cerro, J., San Martin., R. "Mission Plan- ning and Simulation of Unmanned Aerial Vehicles with a GIS-based Frame- work" AIAAGuidance, Navigation and Control Conference and Exhibit. Denver, EEUU, 2006.

56. Jun, Myungsoo, and Raffaello DAndrea. ”Path planning for unmanned aerial vehicles in uncertain and adversarial environments.” Cooperative Control: Models, Applications and Algorithms. Springer US, 2003. 95-110.

57.Grotli, Esten Ingar, and Tor Arne Johansen. ”Path planning for UAVs under communication constraints using SPLAT! and MILP.” Journal of Intelligent & Robotic Systems 65.1-4 (2012): 265-282.

58. Grancharova, Alexandra, and Tor Arne Johansen. ”Distributed MPC- Based Path Planning for UAVs under Radio Communication Path Loss Constraints.” Embedded Systems, Computational Intelligence and Telem- atics in sControl. No. 1. 2012.

59.Chi, Ting-Yun, et al. ”Civil UAV Path Planning Algorithm for Considering Connection with Cellular Data Network.” Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on. IEEE, 2012.

60. Goerzen, Kong, Mettler. ”A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance”, J Intell Robot Syst, 2009.

61. S Koenig, M Likhachev. ”D* Lite.” AAAI/IAAI, 476-483.

62. S. Koenig and M. Likhachev, Incremental A*, in Advances in Neural In- formation Processing Systems 14, T. Dietterich, S. Becker, and Z. Ghahra- mani, Eds. MIT Press, 2002.

63.Hoffmann, Gabriel M., et al. "Quadrotor helicopter flight dynamics and control: Theory and experiment." *Proc. of the AIAA Guidance, Navigation, and Control Conference*. 2007

64. Bouabdallah, Samir, and Roland Siegwart. "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor." Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. IEEE, 2005.

65.Sa, Inkyu, and Peter Corke. "System identification, estimation and control for a cost effective open-source quadcopter." *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

66.Regolamento Mezzi Aerei a Pilotaggio Remoto - http://www.enac.gov.it/repository/ContentManagement/information/N122671512/Regolamento_APR_ed.1.pdf

67. Carlone, Luca, et al. "STEPS: PCS results on 1 st Working Prototype." (2010).

68.Scerri, P., Pynadath, D.V., Tambe, M., “Towards Adjustable Autonomy for the Real World,” 2003 Journal of
Artificial Intelligence Research Volume 17 , Issue 1 (July 2002) Pages: 171-228

69.Heger, Frederik W., and Sanjiv Singh. "Sliding autonomy for complex coordinated multi-robot tasks: Analysis & experiments." (2006).

70.Sellner, B., Heger, F.W., Hiatt, L.M., Simmons, R., Singh, S., “Coordinated Multiagent Teams and Sliding Autonomy for Large-Scale Assembly”, Proceedings of the IEEE, Volume 94, Issue 7, pp. 1425 – 1444, July 2006.

71.M. Sauer. “Mixed-Reality for Enhanced Robot Teleoperation”, Dissertation zur Erlangung des naturwissenschaftlichen Doktorgrades
der Bayerischen Julius–Maximilians–Universität Würzburg, 2010.

72.Cacace, Jonathan, Alberto Finzi, and Vincenzo Lippiello. "A mixed-initiative control system for an Aerial Service Vehicle supported by force feedback."*Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014.

73.Snavely, Noah, Steven M. Seitz, and Richard Szeliski. "Photo tourism: exploring photo collections in 3D." ACM transactions on graphics (TOG) 25.3 (2006): 835-846.

74.-------

75.Sakata, Nobuchika, Takeshi Kurata, and Hideaki Kuzuoka. "Visual assist with a laser pointer and wearable display for remote collaboration." (2006).

76.O. Bimber and R. Raskar. Spatial Augmented Reality: Merging Real and Virtual Worlds. A. K. Peters, Ltd., Natick, MA, USA, 2005

77.Falcao, Gabriel, Natalia Hurtos, and Joan Massich. "Plane-based calibration of a projector-camera system." *VIBOT master* 9.1 (2008): 1-12.

78.C. Harrison, H. Benko, and A. D. Wilson. Omnitouch: Wearable multitouch interaction everywhere. In Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST’11, pages 441–450, New York, NY, USA, 2011. ACM.

79. Y. Hosomizo, D. Iwai, and K. Sato. A flying projector stabilizing image fluctuation. In Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on, pages 31–32, Oct 2014.

80. T. Maeda and H. Ando. Wearable scanning laser projector (wslp)for augmenting shared space. In ACM SIGGRAPH 2004 Sketches,SIGGRAPH ’04, pages 109–, New York, NY, USA, 2004. ACM

81.L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 2992–2997, 2011.

82.V. Milanovic, G. Matus, and D. McCormick. Gimbal-less monolithic silicon actuators for tip-tilt-piston micromirror applications. Selected Topics in Quantum Electronics, IEEE Journal of, 10(3):462–471, May 2004.

83.C. S. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. In Proceedings of the 3rd International Conference on Ubiquitous Computing, UbiComp ’01, pages 315–331, London, UK, UK, 2001. Springer-Verlag.

84.M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In ICRA Workshop on Open Source Software, 2009.

85.J. Scheible, A. Hoth, J. Saal, and H. Su. Displaydrone: A flying robot based interactive display. In Proceedings of the 2nd ACM International Symposium on Pervasive Displays, PerDis ’13, pages 49–54, New York, NY, USA, 2013. ACM.

86.B. Schwerdtfeger, D. Pustka, A. Hofhauser, and G. Klinker. Using laser projectors for augmented reality. In Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology, VRST ’08, pages 134–137, New York, NY, USA, 2008. ACM.

87.K. Tajimi, K. Uemura, Y. Kajiwara, N. Sakata, and S. Nishida. Stabilization method for a hip-mounted projector using an inertial sensor. In 20th International Conference on Artificial Reality and Telexistence (ICAT), Adelaide, Australia, 2010. IEEE Computer Society

88.D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. Visualization and Computer Graphics, IEEE Transactions on, 16(3):355–368, May 2010.

89.A. Wilson, H. Benko, S. Izadi, and O. Hilliges. Steerable augmented reality with the beamatron. In Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology, UIST ’12, pages 413–422, New York, NY, USA, 2012. ACM.

90. A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, “Vision-based hand pose estimation: A review,” Comput. Vision Image Understanding, vol. 108, no. 1–2, pp. 52–73, Oct. 2007.

91. A. D. Wilson, S. Izadi, O. Hilliges, A. Garcia-Mendoza, and D. Kirk, “Bringing Physics to the Surface,” in Proceedings ACM Symposium on User Interface Software and Technology, 2008, pp. 67–76.

92. A. Wilson, H. Benko, S. Izadi, and O. Hilliges, “Steerable augmented reality with the beamatron,” in Proceedings of the 25th annual ACM symposium on User interface software and technology, 2012, pp. 413–422.