DESIGN AND IMPLEMENTATION OF A ROBOT MANAGEMENT FRAMEWORK AND MODULAR GNC FOR ROBOTIC SPACE EXPLORATION

Andrea Biggio [(1)], Carmine Ianni [(1)], Sandro Torelli [(2)], Alessandro Sperindé [(2)], Enrico Simetti [(2)], Federico Salvioli [(3)], Luca Vercellino [(3)], Basilio Bona [(3)]

[(1)] Thales Alenia Space Italia; Strada Antica di Collegno 253, 10146 Torino (TO), Italy;
Email: andrea.biggio@thalesaleniaspace.com, carmine.ianni-somministrato@thalesaleniaspace.com
[(2)] DIBRIS, Università degli Studi di Genova, Via all'Opera Pia 13, 16145 Genova (GE), Italy;
Email: sandro.torelli@dibris.unige.it, alessandro.sperinde@dibris.unige.it, simetti@dibris.unige.it
[(3)] DAUIN – Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino (TO), Italy;
Email: federico.salvioli@polito.it, luca.vercellino@polito.it, basilio.bona@polito.it

ABSTRACT

In the last decade, the robotics field experienced overwhelming increase of robots complexity and spread of robotic applications in consumer, industrial and scientific domains. Software components reuse became a key factor to increase quality of robotic assets and to reduce production time and costs. Thus, the robotics community endeavoured to develop modular robot software architectures, aiming to support the development and integration of robot software modules. Those scattered efforts resulted in a number of products with different features, but a comprehensive product for space applications is yet far to be available.
This paper presents the new TAS-I Robot Management Framework and robot modular architecture, including software modules and algorithms implementing the capabilities for continuous traverse, sample canister acquisition and return and to demonstrate them in outdoor environment. Finally, the future works, including porting to flight-representative hardware, modules optimization and customization for specific tasks, projects and future missions will be discussed.

1. NEED FOR A ROBOT MANAGEMENT FRAMEWORK

Space exploration is one of the humanity greatest endeavours, where robotic technologies are playing a key role in enabling the achievement of more and more demanding mission requirements and ambitious science objectives. While robotic exploration is a pioneering field open to innovative technologies and solutions, the need of common solid foundations to manage the increased complexity of robotic systems is arising, similarly to what happened in the industrial and academic domains. Therefore, great efforts have been spent in the development of modular robot software architectures (aka robot control operative systems, software frameworks, robotic middleware and so on), with peculiar objectives and features [1, 2, 3]. Even in this wide solutions portfolio plenty of de-facto standards like Robot Operative System (ROS), it is impossible to find a comprehensive product (or a combination of them) which may comply with space domain requirements in terms of reliability, availability, maintainability and safety (RAMS).
From these assumptions and the previous experience on robotic frameworks [4, 5, 6] TAS-I pursued the development of its own Robot Management Framework trying to overcome the limitations of the existing approaches.

2. TAS-I ROBOT MANAGEMENT FRAMEWORK OVERVIEW

TAS-I Robot Management Framework stands on the following four pillars:

- **Abstraction:** to provide abstraction from underlying Hardware and Operating System (OS) to ease portability on different robotic platforms;
- **Foundation:** to provide a set of common reliable building blocks which can be reused across various applications;
- **Openness:** to provide open interfaces to ease software modules integration;
- **Focus:** to provide high-level Application Programming Interface (API) to ease robot software modules implementation.

To achieve these objectives TAS-I designed the Robot Management Framework architecture depicted in Fig. 1.

3. ROBOT LAYER

The Robot Layer consists in the physical robotic platform, including hardware and software components such as sensors, actuators, on-board-computer(s), interfaces, drivers and the operating system where the architecture is deployed. The architecture has been designed to be independent from the robotic platform, enabling reuse in different projects.

4. WORKFRAME LAYER

The WorkFrame Layer is the core of the Robot

Management Framework, implementing all the common reliable building blocks that may be reused across. It is based on the work presented in [5] and includes additional functionalities:

- **KAL:** Kernel Abstraction Layer
- **HAL:** Hardware Abstraction Layer
- **WFCore:** WorkFrame Core Functionalities
- **NET:** Network Communication
- **BBS:** BlackBoard System
- **CMAT:** Mathematical Library

This layer exposes a set of high-level APIs to manage the different functions that are not available as open source to ensure the reliability of the system.

4.1. Kernel Abstraction Layer

This layer is the only one directly interfacing with the underlying OS. The goals of this layer are:

- to encapsulate the OS resources APIs in easier to use classes;
- to provide objects to access I/O devices;
- to handle the creation of tasks;
- to deal with the scheduling of the system;
- to guarantee to portability of the higher levels.

This last objective affects all of the above ones, as in order to guarantee the portability, all the classes have to provide unique interfaces independently of the underlying operating system. Even if [5] supports many OS, current TAS-I robot management framework implementation supports Linux, Linux/RTAI and RTEMS. As the OS system calls cannot always be mapped one another, the KAL abstracts only the common features. Direct use of OS-specific calls is not forbidden but the porting of such software pieces has to be done manually.

4.2. Hardware Abstraction Layer

This layer provides a general way to interact with the robot sensors and actuators. Thanks to this layer the user tasks are device independent. When the hardware changes, just the portion of HAL relevant to that particular component(s) has to be re-coded according to the defined software interface, leaving the higher layers unchanged.
Similarly it is possible to implement HAL drivers to perform dry-run tests without hardware in the loop (e.g. by communicating with a simulator or providing dummy data).

4.3. WorkFrame Core

This library contains the core functionalities of system management, which includes the handling of the system state, the proper scheduling of all tasks and control over the deadlines, the creation and request of resources and the general task functioning.
In order to offer all the services, a set of system tasks is included in the WorkFrame Core library:

- **System Manager**, which controls the evolution of the system state, and the unique handler of all the requests for resources or tasks;
- **Scheduler**, to execute and monitor the scheduling of user tasks and modules;
- **Debug Console**, managing framework and user tasks debug messages.
- **WorkFrame Console**, to control basic functionalities (e.g. start and stop the WorkFrame)
- **Watchdog**, to monitor the state of WorkFrame and user tasks state.

4.4. Network Communication

The Network Communication library contains all the

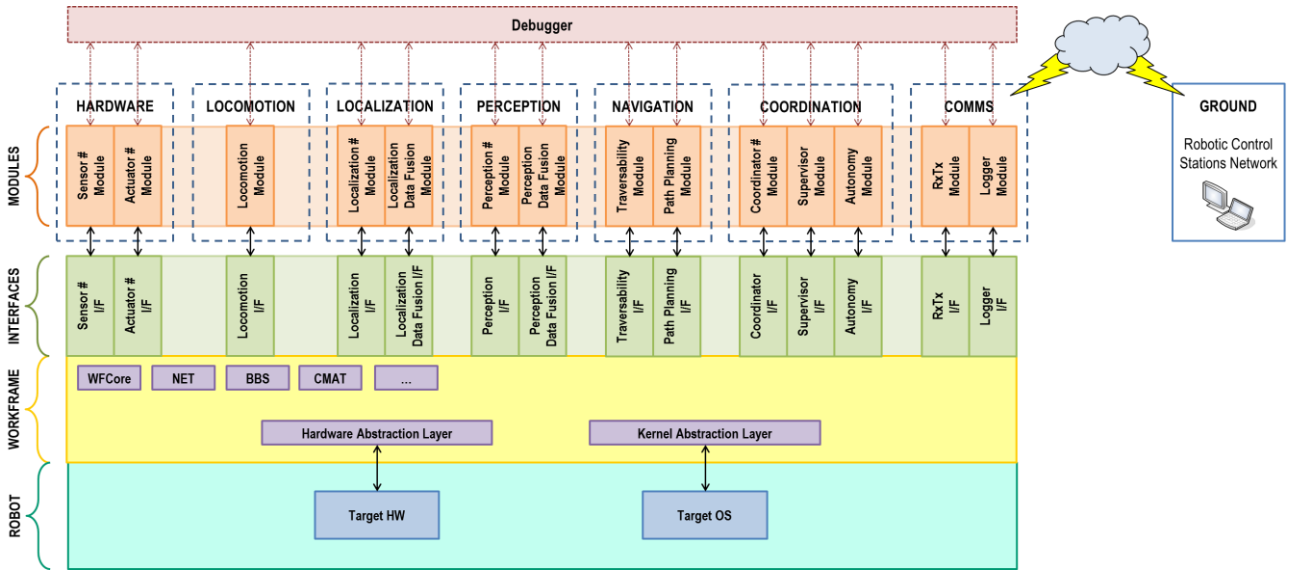

Figure 1. Robot Management Framework Architecture

classes and functions needed to perform remote communications.
The Client/Server paradigm is implemented by dedicated classes, easing the communication with remote modules and external systems. In the same way, it is planned to implement the Publish/Subscribe paradigm.
At the moment the Ethernet and Bluetooth medium are supported, while CANbus is under testing.
The NET library does not preclude the possibility of communicating with other Robotic Middleware; indeed it is possible to use other communication libraries and connect the modules to e.g. a ROS node.

4.5. BlackBoard System

The BlackBoard system implements the data sharing mechanism between tasks, providing high-level APIs to exchange data in a thread-safe way using the *BlackBoard* a shared memory segment containing both the data and a header part used internally by the classes of the BlackBoard System. Three dedicated tasks are assigned to the management of the BlackBoard system:

- the **BlackBoard Manager**, which creates the BlackBoard, handles the declaration of all data and creates the resources to implement mutual exclusive access;
- the **Execution Manager**, which manages the access to the resources by the user tasks;
- the **Connection Manager**, which takes care of interface with all the remote tasks exploiting the services of the NET library.

4.6. Mathematical Library

This library implements a minimal set of classes to handle matrices and vectors, with particular methods, very often used in robotics, for operation like Singular Value Decomposition, to transform Roll-Pitch-Yaw vectors to their corresponding Rotational matrices and vice-versa, and so on.

5. INTERFACE LAYER

The Interface Layer implements standardized means enabling communications between modules and to the system. It gives the needed openness to the framework in order to allow collaborative design, development, integration and testing of software modules. In particular it provides the following features:

- **Data Interface**, to ensure data types consistency;
- **Messages Passing System**, to ease messages exchange between modules.

5.1. Data Interface

The Data Interface provides a uniform access to the BlackBoard that contains the data exchanged by all the modules. The APIs take care about the concurrent access to the memory and embedded serialization ensures the data consistency among different systems.
Alongside to the provided APIs, the users can easily define and maintain the data structures that will be exchanged on the BBS and over the network.

5.2. Messages Passing System

The Robot Management Framework architecture is based on the exchange of standardized messages. The messages are used to give specific commands to the modules and receiving acknowledgments. A timestamp is automatically attached to each new message that is generated.
The architecture defines three different messages queues

- An **Emergency Queue**, which is used by the for high priority messages;
- A **Debug Queue**, which is used to transmit commands that should be only executed in a debug context;
- A **Nominal Queue**, which is the nominal communication channel.

The MPS APIs ease messages generation and processing, moreover it automates messages handling, including the verification status acknowledgements. It implements also a message relay function, enabling the passing of a message to a recipient passing through one or more relay modules.

6. MODULES LAYER

The Modules Layer is built on top of the architecture, which defines only the base module class. From the base module class is it possible to derive customized subclasses implementing the modules functionalities.
To ease the modules development, the modules layer exploits the underlying APIs and a series of high-level APIs specifically for:

- **Finite State Machine Definition**, to implement modules FSM and relevant functions;
- **Modules Scheduling**, to implement periodic and aperiodic modules;
- **Modules Commands**, to automate commands parsing and related functions;
- **Modules Configuration**, to ease and standardize modules configuration;
- **Modules Modes**, to implement different behaviour according to the operational contexts;
- **Remote Modules Implementation**, to distribute modules while still natively communicating with the framework core.

6.1. Finite State Machine Definition

The Module class provides a set of APIs to build and manage its own FSM. Each defined state binds a function that is executed as “state function” at each module cycle.
The architecture defines also the concept of meta-state:

a pointer to a given state which can be changed at runtime. This way it is possible to associate different state functions to the same state name at runtime, changing the module behaviour. This concept allows all the other modules to be unaware of how many different possibilities exists for each meta-states, since they will refer just to them and not the specific implementation alternatives. Furthermore, this increases the flexibility during execution, since without restarting the robot different alternatives can be quickly selected.

6.2. Modules Commands

Each Module can bind a command string to a specific function which implements the command. This way the code results more modular, as each command function (as-well-as state/meta-state functions) are separate and easy to be maintained. If an unknown command is received, the error is reported to the module which issued the command. Other errors can be defined by the user and have to be handled according to the desired Failure Detection Isolation and Recovery implementation.

6.3. Modules Scheduling

Each Module can be configured either as a periodic or as an aperiodic module. In the periodic mode, the module will run at fixed time intervals, executing its state function once per cycle. In the aperiodic configuration, the module will poll the messages queues to execute the commands issued by other modules.

6.4. Modules Configuration

Each module has its own configuration file, containing at least the set of parameters needed by the framework (e.g. periodic task sample time, default operative context, commands list). Moreover, the configuration file can host module-dependant parameters (e.g. algorithm parameters).
Configuration files are read at start-up and can be managed in user-defined module states (e.g. a config state where a user can change configuration parameters and save a new configuration file) providing on-the-fly modules configurability.

6.5. Modules Execution Context

The architecture defines two parameters that can be set for each module to run in a different execution context.
The first parameter sets if the module runs in a **Dry-Run** or **Operative** context. The Dry-Run context is intended to allow the user testing the module in specific conditions (e.g. a sensor module that generates data according to the sensor model). The Operative context is the nominal case, where the user want to run the module as-is (e.g. a sensor module that acquires data from a real sensor).
The second parameter sets if the module runs in a **Debug** or **Mission** context. The Debug context is intended to let the user define specific debug execution procedures, separated from the nominal Mission context. In the Debug operative context the Debug Commands Queue is active and so the modules accept debug commands which can be used to trigger specific behaviours (e.g. inject a failure).
As the two parameters are handled separately, the Execution Context can be selected among the following:

- **Operative – Mission:** which is the nominal context.
- **Dry-Run – Mission**, where the modules can perform their nominal work in dry-run mode;
- **Operative – Debug,** where the modules can be debugged in their operative conditions;
- **Dry-Run – Debug**, where the modules can be debugged while running in Dry-Run mode.

Those configuration parameters can be either changed at runtime, if the behaviour is implemented by the user (e.g. in a configuration state of the module FSM).
The user who desires to exploit these features has to implement the different behaviours inside the Module FSM, exploiting the APIs enabling this design paradigm.

6.6. Remote Modules

To distribute the computational effort on multiple machines, it is possible to implement Remote Modules which run on different hosts that the one which runs the Robot Management Framework.
Those remote machines needs to be compatible with the Robot Management Framework, so that the Remote Modules running on them can link the needed framework libraries. Apart from this effort, setting a module as remote is matter of changing its initialization parameters, including network IP address and port of the machine where the core system is running.
Other function, such as the data exchange to the BlackBoard system are managed through the same APIs, so the rest of module code remains unchanged.

7. MODULAR ROBOT CONTROL SOFTWARE ARCHITECTURE

Over the Robot Management Framework, a Modular Robot Control Software (RCS) architecture has been designed. As described in the previous chapters, the basic building block is the Module, which can be specialized and replicated to design and implement a generic RCS (e.g. rover GNC).
Five module types have been defined:

- **Sensor**, to interface with robot sensors;
- **Actuator**, to interface with robot actuators;
- **Resource**, these modules implement the RCS functions and algorithms, process Sensors Modules data, provide input to the Actuator Modules;
- **Coordination**, in charge of coordinating a set of

Resource Modules to implement complex tasks;
- **External**, implementing non-core functions of the RCS like Logging and Debugging.

Alongside the module types, which can be used to define the RCS structure, an overall FSM concept has been defined to provide a baseline implementation of the behaviour of such modules (Fig. 2):
- **Startup**, in this state the Robotic Framework tasks are started and all the local and remote modules register to the system;
- **Config**, this state allows the configuration of the modules, it is possible e.g. to load/save configuration files, change module dependent parameters and set the module meta-states. In this state it is possible to select each module execution context;
- **Standby**, this state is a transition state where the system waits for a system command to start the execution (so switching to the hold state) or to shut down (so switching to the Shutdown state). In this state the sensors and actuators can be switched on and off;
- **Hold**, this state is and execution state where the module is idle and the sensors and actuators are switched on;
- **Run**, this is the module execution state;
- **Shutdown**, in this state the shutdown procedure is executed;
- **Safe**, this state is an off-nominal state where a module enters in case of failure. From this state it is possible to analyse the module errors and decide upon switching back to standby and perform a module re-configuration, to restore the execution or shutdown the system;

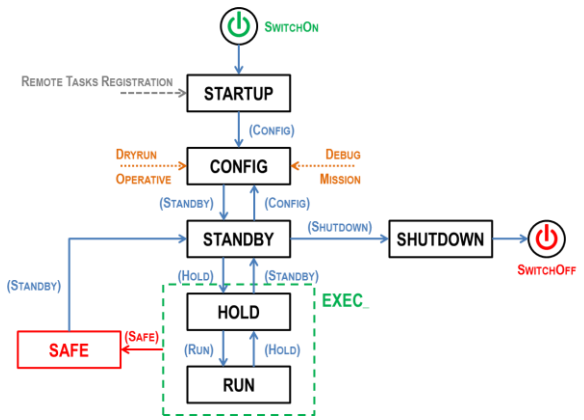

Figure 2. Finite State Machine

Obviously this FSM provides a concept of module states, but state/meta-state functions bond to each state has to be implemented by the user according to the desired behaviour. Other Robot management Framework users may define their own FSM as well.

8. MODULAR GNC IMPLEMENTATION

The modular RCS architecture has been tailored for TAS-I research rovers, implementing the capabilities of a sample canister acquisition mission to be demonstrated in outdoor TAS-I ROvers eXploration facilitY [7]. The baseline mission scenario is divided in three phases:
- **Sample Canister Identification**, by looking at the rover Tracking Camera pictures, the sample container is identified and manually selected by a human operator;
- **Rover Traverse**, a tracking algorithm estimates the selected sample container position and provide it as goal to the rover GNC. Then the rover automatically approaches the target;
- **Sample Canister Acquisition**, Once in the neighbourhoods of the sample canister, the robotic manipulator is commanded to acquire the sample canister, using the visual feedback to approach the canister interface.

The tailored Robot Control Software hierarchy is depicted in Fig. 3.
The **Autonomy** module is the intelligence of the robotic system and the one that implements the highest level of FDIR. It is in charge of decomposing the plan submitted from ground into commands sequences to be executed by the lower level modules.
The **Coordinator Navigator** module coordinates the resource modules and other Coordinator modules to implement stop-and-go and continuous GNC chains.
The **Coordinator Manipulation** module is in charge of managing the coordination between the Manipulator and the Target Racking modules to implement vision-based object manipulation.
The **Coordinator VisualTracking** module implements the needed coordination between the Target Tracking modules and the PTU module to chase the selected target.
The **PerceptionDataFusion** module is in charge of merging the local DEMs generated by the Perception modules with the global DEM, building a consistent map of the explored area. It coordinates also the PTUs actuation to perform the needed scans with perception sensors.
The **LocalizationDataFusion** module is in charge of merging Localization modules output to provide the 6D pose estimation of the robot.
The **LocalizationVisual/Mechanical/Inertial** modules implement algorithms to calculate robot pose estimation based on vision/encoder/inertial sensors.
The **PerceptionToF** module implements algorithms to generate local DEMs based on ToF camera sensor input. A PerceptionVisual module generating local DEMs based on Head Camera stereo bench is available but not included in the present architecture.
The **Traversability** module implements the algorithms

that generate the Navigation Map (NavMap) starting from the global DEM and the trafficability parameters (e.g. maximum traversable slope, maximum traversable step, maximum steering angle) of the robotic platform.
The **Path Planning** module implements algorithms to generate a safe and optimal trajectory on the NavMap, given a navigation goal.
The **Locomotion** module implements the locomotion equations which translate the trajectory into speed and jog values to the LocomotionActuator module.
The **TrackingTarget** module implements an algorithm to estimate the 3D position of a selected Region of Interest (ROI) in an image.
The **TrackingMarker** module implements an algorithm to estimate the 6DoF pose of one or more marker tables (a-priori known set of markers).
The **ManipulatorControl** module implements the commands sequences to control a manipulation system (e.g. a robotic arm).
The **Logger** external module is in charge of monitoring, logging (e.g. to file) and distributing (e.g. to a Ground Control Station) modules parameters (e.g. module status, exchanged data) at a configurable frequency.
The **Debugger** module is in charge of injecting specific debug commands to trigger particular behaviours while debug mode is active.
As many of the enlisted modules functions have been developed in the frame of [4] and related projects, only the new modules and the ones with relevant upgrades will be presented in the next paragraphs.

8.1. Vision-Based Guidance

The modular GNC system features visually assisted navigation capabilities that allow to automatically approach objects or areas of interest (later addressed as targets) in the visible surroundings of the robot. A graphical user interface lets the operator to see the on-board stereo camera video feed and select a rectangular ROI bounding the target.
The Visual Target Tracking module implements an algorithm which processes the stereo images pairs and provides the ROI position in the three dimensional space to the rest of the GNC.
The module processes incoming data in two main steps: a *tracking* step followed by a *target localization* step. The *tracking* step relies on the OpenTLD implementation of the Track Learn Detect algorithm and allows to track a desired target in a single camera image (one of the stereo camera eyes). The *target localization* step makes use of the stereo image pair to build the disparity map and consequently a point cloud of the perceived surroundings. The points inside the tracked area are clustered using k-means in background and foreground points whose centroid is used as a position estimation of the target.
The Visual Tracking module has been implemented as a remote module of the architecture to provide it dedicated computational resources.
The higher level GNC modules exploit the Visual Target Tracking module functionalities through four commands with different level of autonomy:

- **Point and locate:** this command allows the operator to select a target, make the camera to point towards it and know where the target is located with respect to the robot.
- **Point, locate and approach:** this command is behaving as the previous except that it forwards the target's location information to the rest of the GNC to make the robot approach the target within a user specified boundary.
- **Point, locate, approach and track:** this command loops the last procedure until the robot is inside a user specified maximum distance from the target keeping the target tracked and pointed. This is useful when following a moving object or when far away target are to be approached so an iterative refinement of the first uncertain estimation is required.

8.2. Digital Elevation Map Generation and Fusion

To serve the purpose of continuous traverse, the DEM generation is based on Time-of-Flight (ToF) camera environment perception. The use of ToF cameras, which native output is the 3D Point Cloud of the perceived

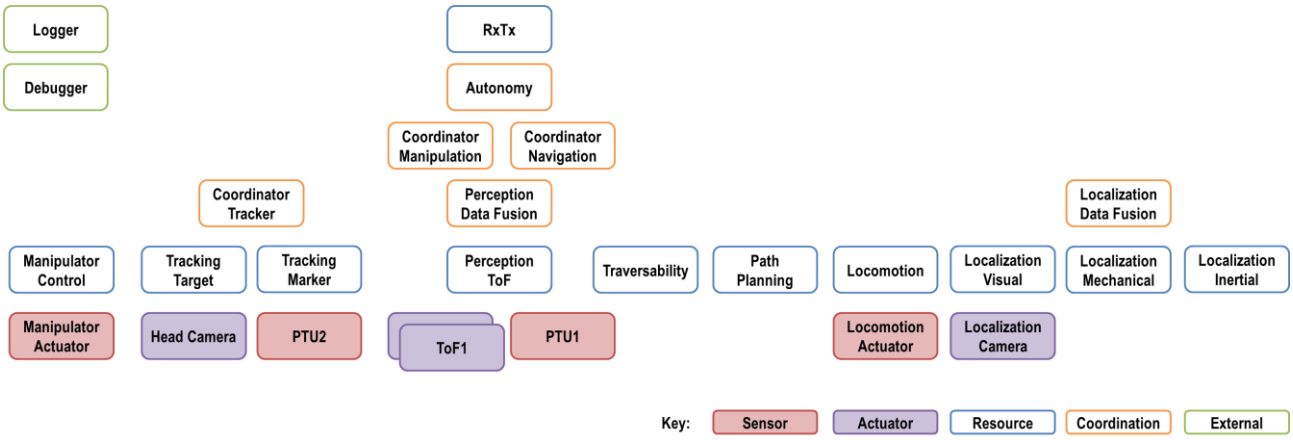

Figure 3. Modular GNC Hierarchy

surroundings, avoid the need of performing stereo processing resulting in an order of magnitude speed-up of 3D perception [4].
Commercially available ToF cameras Horizontal Field of View is too narrow to perceive enough terrain in front of the rover. Therefore a double ToF sensor assembly has been integrated on the rovers, providing a larger field of view while keeping the PTU pointing in the rover motion direction at a constant tilt angle during traverse.
DEM generation sequence is triggered by the Navigator module, which is in charge of orchestrating the resource modules devoted to the traverse execution.
ToF cameras raw data are merged and filtered by the perception modules to obtain the local DEM, which is then used to update the global DEM by the perception data fusion module.
Depending on the robot size and configuration, a blind area is projected around the rover. To mitigate issues both while merging DEMs and while generating Navigation Maps, a filling algorithm has been implemented. This algorithm analyses the blind area edge and interpolates the values to generate a smooth surface that completes the DEM.

8.3. Visual Odometry

The Visual Odometry module is in charge of the robot 6D pose estimation. Its implementation is based on the open source Libviso2 library [8, 9]. Libviso2 is a visual odometry library that allow to estimates the movements of the robot from images just using a single CPU and at a rate suitable for robot localization.
The Visual Odometry module has been implemented as a remote module of the architecture to provide it dedicated computational resources.
The module acquires stereo image pairs from the localization camera, rectifies them and computes the motion estimation between two temporally consecutives image pairs. Once the visual odometry module has computed an estimate, it sends it to the task visual localization module using the network layer. This secondary module receives estimates and periodically sends them, always using the network layer, to the remote robot framework in a coherent format.
Since the visual odometry system is executed on a secondary computer and the other sensors acquisitions and the localization data fusion is made on the primary one, that is the computer where the robot framework is, a way to synchronize computers is required. In practice, the correlation between remote measurements and the local ones exploits a common pulse generated by a Differential GPS system (which is used only as ground truth for robot trajectory data analysis). Both the computers are physically connected to the DGPS system and at each image acquisition the visual odometry module stores the related timestamp to be later sent with the corresponding estimate. This way when the robot framework receives estimates it is able to temporally trace and merge them with the other sensors measurements, compensating possible computational and network delays between the time when the stereo pairs are acquired and the received estimate by the framework.
The module offers a graphical user interface too, allowing the users to quickly see the current robot position and orientation, the trajectory followed and to easily change some parameters such as the image acquisition source, camera parameters and visual odometry related parameters.

8.4. Manipulation System with Visual Servoing

The acquisition and storage of the sample canister is performed by means of a 6DoF robotic arm with an integrated clamping end effector. The robotic arm is controlled by the Manipulator Control module, which exploits the direct and inverse kinematics control provided by an external control library developed by the authors. The sample canister mock-up is a simple box with an aluminium handle, where a black and white marker pattern has been applied to estimate the 6DoF pose of the handle and close the visual servoing loop.
The visual pose estimation algorithm makes use of the Aruco open source library, based on OpenCV. The library provides a handily API to build, detect and track marker patterns.
To enable the interaction of the rover with multiple objects in view of more complex operations (e.g. inspection, maintenance, localization with a fixed object or another robot), an object data-base has been developed, while the Tracking Marker module implements the capability of tracking multiple marker patterns at once.

9. CONCLUSIONS

TAS-I Robot Management Framework has been developed and validated. Functional modules are ready to be ported on the new architecture after algorithms testing and characterization.
The mission scenario is going to be demonstrated as part of the STEPS2 project test phase, where a characterization of the major building blocks and algorithms is on-going [7].

10. FUTURE WORKS

The Robot Management Framework is a living project of TAS-I which pursues the TRL rising in the next years. It is planned to upgrade the framework with:

- the integration of CANbus and DTN support in NET library;
- the integration of manipulator control library within the Framework Layer;
- the upgrade of RTEMS KAL and testing on flight representative avionics;

- the consolidation of the overall framework architecture implementation and documentation according to the ECSS standards.

From the GNC standpoint, lessons learned from STEPS2 project testing phase (still to be completed) will be implemented. The integration of a science autonomy algorithm to automatically locate the sample canister can be pursued too in the next future.
The interaction with the environment based on marker detection and tracking is going to be further improved to enable more complex operations, such as rover ranging and rover localization with respect to a fixed object (e.g. a lander mock-up).
Finally, TAS-I Robot Management Framework has already been selected by ESA for the Eurobot Ground Prototype [10] restructuring and modularization, to increase the robot reliability and reusability in view of planned METERON (Multi-Purpose End-To-End Robotic Operation Network) experiments and future human-collaborative robotics research activities.
TAS-I is pursuing the use of Robot Management Framework in post-exomars Mars exploration missions (e.g. Mars Sample Return Sample Fetching Rover) and Lunar exploration rovers (e.g. Lunar Volatiles Prospector).

11. ACKNOWLEDGMENTS

The activities subject of this paper have been performed in the frame of STEPS program - Systems and Technologies for Space Exploration - a research project co-financed by Regione Piemonte (Piedmont Region) within the Phase 2 of P.O.R. - F.E.S.R. 2007-2013 EC program.

12. REFERENCES

1. Elkady A., Sobh T. (2012), Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics, Volume 2012.*
2. Calisi D., Censi A., Iocchi L., Nardi D. (2008), OpenRDK: a modular framework for robotic software development. *The 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems.*
3. Joyeux S., Schwendner J. (2014), Modular Software for an Autonomous Space Rover. *i-SAIRAS 2014.*
4. Biggio A., Merlo A., Tramutola A. (2012), Test Bench for Robotics and Autonomy: Advancements in Navigation for Space Exploration. *i-SAIRAS 2012.*
5. Simetti E. et al (2011), A new software architecture for developing and testing algorithms for space exploration missions. *Intelligent Service Robotics volume 4(2)*, Springer-Verlag, pp135-146.
6. Simetti E. et al (2010), A Portable Object Oriented SW Framework for Real-Time Control of Robot and Multi-Robot Systems. In *Control Themes in Hyperflexible Robotic Workcells* (Eds. F.Basile, P.Chiacchio), CUES, pp129-143.
7. Biggio A. et Al. (2015), Validation and Verification of Modular GNC by means of TAS-I Robot Management Framework in outdoor ROvers eXploration facilitY. *ASTRA 2015.*
8. Kitt B., Geiger A., Lategahn H. (2010), Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. *Intelligent Vehicles Symposium 2010.*
9. Geiger A., Ziegler J., Stiller C. (2011), StereoScan: Dense 3D Reconstruction in Real-time. *Intelligent Vehicles Symposium 2011*.
10. Medina A., Pradalier C., Paar G., Merlo A., Ferraris S. (2011), A servicing rover for planetary outpost assembly. *ASTRA 2011*.