# A COMMON DATA FUSION FRAMEWORK FOR SPACE ROBOTICS: ARCHITECTURE AND DATA FUSION METHODS

**Raùl Domínguez [(1)], Romain Michalec [(2)], Nassir W. Oumer [(3)], Fabrice Souvannavong [(4)], Mark Post [(2)], Shashank Govindaraj [(5)], Alexander Fabisch [(1)], Bilal Wehbe [(1)], Jérémi Gancet [(5)], Alessandro Bianco [(2)], Simon Lacroix [(6)], Andrea De Maio [(6)], Quentin Labourey [(6)], Vincent Bissonnette [(4)], Michal Smíšek [(3)], Xiu Yan [(2)]**

*[(1)] DFKI GmbH Robotics Innovation Centre, Robert Hooke Straße 1, Bremen, Germany*
*raul.dominguez|bilal.wehbe|alexander.fabisch@dfki.de*
*[(2)] University of Strathclyde, 16 Richmond Street, Glasgow G1 1XQ, United Kingdom*
*mark.post|romain.michalec|alessandro.bianco|xiu.yan@strath.ac.uk*
*[(3)] DLR, Münchener Straße 20, 82234 Weßling, Germany*
*nassir.oumer|michal.smisek@dlr.de*
*[(4)] Magellium SAS, 24 rue Hermès, 31521 Ramonville-Saint-Agne, France*
*fabrice.souvannavong|vincent.bissonnette@magellium.fr*
*[(5)] Space Applications Services NV, 325 Leuvensesteenweg, 1932 Zaventem, Belgium*
*jeremi.gancet|shashank.govindaraj@spaceapplications.com*
*[(6)] LAAS-CNRS, 7 avenue du Colonel Roche, 31031 Toulouse, France*
*simon.lacroix|quentin.labourey|andrea.demaio@laas.fr*

ABSTRACT

Data fusion algorithms make it possible to combine data from different sensors into symbolic representations such as environment maps, object models, and position estimates. The software community in space robotics lacks a comprehensive software framework to fuse and contextually store data from multiple sensors while also making it easier to develop, evaluate, and compare algorithms. The InFuse consortium, six partners in the industrial and academic space sector working under the supervision of a Program Support Activity (PSA) consisting of representatives from ESA, ASI, CDTI, CNES, DLR, UKSA, is developing such a framework, complete with a set of data fusion implementations based on state-of-the-art perception, localization and mapping algorithms, and performance metrics to evaluate them. This paper describes the architecture of this Common Data Fusion Framework and overviews the data fusion methods that it will provide for tasks such as localisation, mapping, environment reconstruction, object detection and tracking.

## 1. INTRODUCTION

In the context of perception, localization and mapping in space robotics, common evaluation and deployment frameworks are crucial for efficiently developing reliable robotic solutions. Qualification of software for space is indeed highly demanding. Methods and tools that make it easier to develop software for space and evaluate it as early as the prototype stages are highly valuable. This is the motivation of the European Commission's Horizon 2020 Strategic Research Cluster in Space Robotics Technologies, which comprises several projects to develop open, modular and reusable solutions in the domains of Robotic Control Operating Systems (RCOS) (Operational Grant 1, OG1) [1][2], autonomy (OG2) [3], perception and localisation (OG3) [4], and sensors (OG4) [5].

We are currently developing a software framework to implement, evaluate, and deploy data fusion algorithms for applications such as localisation, mapping, environment reconstruction, and object detection and tracking: a Common Data Fusion Framework (CDFF). We designed it to handle the challenges of developing and integrating sensor data fusion algorithms in a space context: (1) it will be compliant with the requirements that space grade software impose at interface level, and partially conform to lifecycle and coding guidelines based on ECSS-E-ST-40C standards (European Cooperation for Space Standardisation), (2) it will be easily deployable in ESROCOS [1][2] that uses TASTE modeling [6], (3) it will be experimentally validated by our porting of certain Data Fusion Nodes (DFNs) and Processing Compounds (DFPCs) to RTEMS, and (4) it will be validated in a partially hardware-accelerated setup, with some data fusion processing taking place on a FPGA coprocessor (e.g. Xilinx Zynq SoC).

Although our framework targets space robotics, one of our design guidelines is also to facilitate its integration with RCOSes other than ESROCOS, in particular with ROCK (Robot Construction Kit) and the widely-used ROS (Robot Operating System). Furthermore, it comes with tools that make it possible to evaluate perception and localisation modules with framework-independent

logged data and without using any RCOS. These two points make it RCOS-independent and therefore suitable for many terrestrial applications with minimal modification. Furthermore, the CDFF after its initial development will be available for the general public as open source project.
Section 2 describes the architecture of the framework, and presents its main conceptual components and the corresponding software modules. Section 3 lists the data fusion methods that will be released with the CDFF. Section 4 describes the experimental validations that will demonstrate the usefulness of the framework, and Section 5 concludes the paper.

## 2. ARCHITECTURE OF THE CDFF

Data Fusion Nodes (DFN), defined as atomic processing units that perform a single data fusion function, constitute the core of the CDFF. Their atomicity makes them reusable and specialised. Consequently, they need to be connected and coordinated to each other in order to produce a particular data fusion product. We call a particular arrangement of DFNs, together with the controller that coordinates them, and the local data store (if any) for the data they use or generate, a Data Fusion Processing Compound (DFPC). Activation and deactivation of these DFPCs, as well as all other control and data flows within the framework, are the responsibility of an Orchestrator component. Finally, the last component of the framework is a Data Product Manager that stores and retrieves data from persistent memory on request from the Orchestrator. Fig. 1 shows a diagram that summarises this architecture.
We have divided the CDFF into three component groups: (1) CDFF-Core comprises the DFNs, (2) CDFF-Support provides the tools to connect these into DFPCs, coordinate their operation, and manage their data products, and (3) CDFF-Dev is an environment for developing and evaluating data fusion algorithms, independently of the target robotic system and of its RCOS.

### 2.1. Data fusion libraries

A DFN is an atomic processing entity that fulfills a given data fusion function. It is the smallest unit of more a complex task, defined by its function, input and output. However, a DFN can actually be made up of a combination of elementary functions which may not expose their input/output. It presents two control interfaces: (1) `configure()` sets all the configuration parameters of the DFN, then (2) `process()` calls library functions to compute the outputs of the DFN.

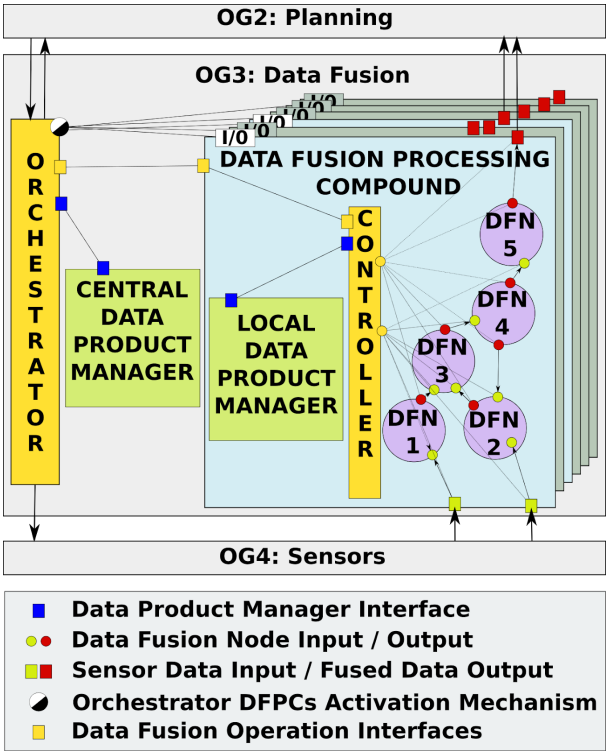



*Figure 1. The Orchestrator manages the queries to the Central Data Product Manager, the activation of different Data Fusion Processing Compounds (DFPCs) and the operating modes of OG4 to fulfill requests from the planning algorithms in OG2.*

The DFN interface has been designed to be compatible with the ESROCOS RCOS. Nonetheless, DFNs do not need to be deployed as single ESROCOS modules. Modules instead can include entire DFPCs or even a complete set of DFPCs, Orchestrator and DPM.
A DFN Template offers an abstraction of the essential characteristics of all DFNs, and allows their efficient management. The DFN Template incorporates the following information: (1) Generic Description, (2) Input(s) and Ouput(s) data, (3) Input(s) Parameters, (4) Estimated performance and cost, (5) External library dependencies, (6) Diagnostic capacities and (7) Unit test. Further details are presented in the deliverables corresponding to the Orbital and Planetary Track Test Plans [7, 8].

### 2.2. Data fusion solutions

CDFF-Support is a set of components designed to run on the target system along with the DFNs. These components provide supporting tools to use multiple DFNs together and in a coordinated fashion to produce a complete solution to a given data fusion requirement.
Furthermore CDFF-Support provides the Data Products Manager (DPM) which stores a consistent

representation of the environment, a history of acquired pre-processed sensor data, estimated poses, and a selection of the generated fused data products, to deliver them under request to OG2. Data is also stored locally within DFPCs so that it can be further exchanged between DFPCs as well as made available for the central DPM. The three main components of CDFF Support are the DFPCs, the Orchestrator, and the DPM. These components are depicted in Fig. 2.

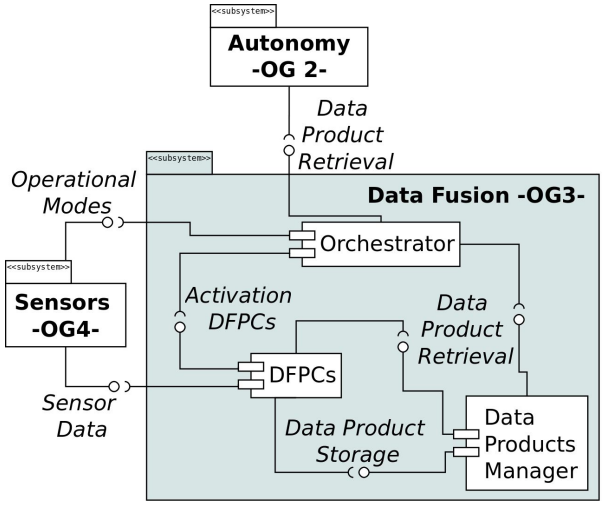



*Figure 2. This diagram presents how the three components of CDFF-Support interact, and also how the CDFF interfaces with OG2 and OG4.*

| DFPC : Lidar Map-Based Localisation | |
|---|---|
| Description of Data Flow | - Input: lidar point cloud, pose estimate<br>- Output: pose estimate<br>- DFNs: PointcloudMatcher, PoseEstimator |
| Data Product Management | - Graph-Map: Pose Graph, Scan Map, Key Frames |
| Description of Control | 1. getLocalMap<br>2. PointcloudMatcher.doICP<br>3. PoseEstimator.estimatePose |

*Table 1. Description of the Lidar Map-Based Localisation DFPC. Similar descriptions will be available for each DFPC released with the CDFF*

Each DFPC is characterised by its function, the data streams that it receives and produces, including the corresponding metadata (for instance timestamps and geometric models), the operations that it can execute on demand, the DFNs it uses and how these are configured and set up. We use description templates such as the one in Tab. 1 to document DFPCs. The fields in this template are: (1) Description of Data Flow, (2) Data Product Management, and (3) Description of Control.

The orchestrator has the main task of receiving queries from the Autonomy Framework (OG2) , activating and providing the fused data products to OG2. It acts as the central coordinator in the target system to control the activation states of DFPCs. The orchestrator has the following functions: (1) Interface between OG2 and OG3, (2) Translate the perception and localisation data into the format required by OG2, (3) Interface with the OG4 Instrument Control Unit (ICU) to configure a limited set of operational modes, (4) Interface with the Data Product Management (DPM) tool to provide mechanisms for querying fused data products and (5) Activation and deactivation of DFPCs according to data product requests and operational modes of the sensors. This last function does not interfere with internal DFPC decision making processes.

The role of the DPM is to handle the selection, structuring and storage of all the data processed or produced by the CDFF that may be re-used, either internally by OG3 processes or to satisfy OG2 requests. Additionally, it is the interface through which robots expose and retrieve the CDFF data products in multi-robot scenarios, and also the interface through which ground operators can access the CDFF data products. The DPM can be seen a robotics-dedicated Geographic Information System (GIS). With respect to the activated DFNs and DFPCs in the CDFF, the DPM processes the data insertion requests. Internally, it manages all the spatial related data by implementing insertion, deletion or update functions, aiming at satisfying future needs for data products and storage constraints.

### **2.3.** Development toolkit

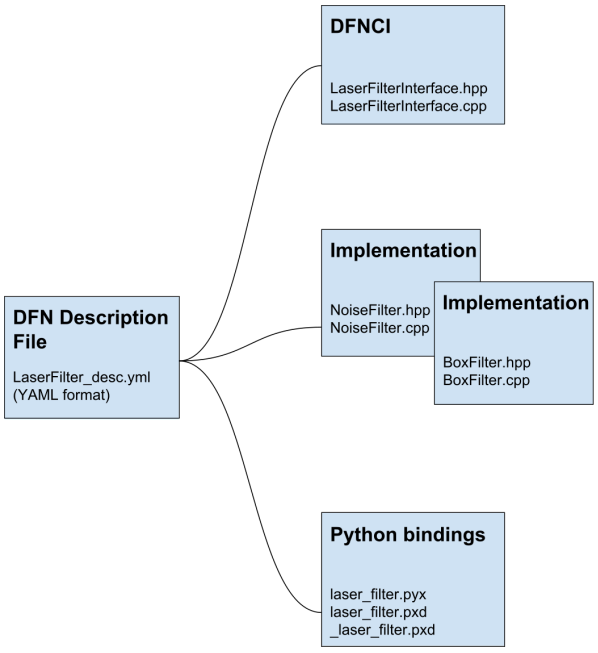
CDFF-Dev provides tools for testing and prototyping data fusion solutions independent of the target RCOS. That includes a tool to replay, visualise and analyze logs in Python, and a framework to develop and test new DFNs that use signal processing or machine learning algorithms, in particular, for data filtering and outlier detection. None of these tools are deployed on the target system. In addition, code generators for DFN and DFPC scaffolds and corresponding Python bindings are provided. They are tools for developers of data fusion algorithms.

The first step to evaluate or implement a DFN or DFPC using CDFF-Dev, is to generate a DFN or DFPC description file from the DFN or DFPC template, described in Sections 2.1 and 2.2. From the description

file, the code of the interface is generated. As an example, Fig. 3 a) shows the DFN description file in YAML format and Fig. 3 b) displays the artifacts that are created by the DFN code generator. The Python bindings for DFNs and DFPCs are generated and bindings for InFuse data types are already provided to make the prototyping of DFNs and DFPCs more convenient in Python.

```
name: LaserFilter
implementations:
    - NoiseFilter
    - BoxFilter
input_ports:
    - name: scanSamples
        type: LaserScan
        doc: samples of a laser scan
    - name: laser2BodyTf
        type: RigidBodyState
        doc: laser frame to body frame
output_ports:
    - name: filteredScans
        type: LaserScan
        doc: filtered laser scans
```

*(a)*




*(b)*

*Figure 3. (a) Example of a DFN description file based on YAML. (b) The DFN code generator creates an abstract C++ base class that defines the interface of the DFN, templates for concrete implementations and Python bindings for these implementations.*

The interfaces of DFNs and DFPCs are kept as minimal as possible to ease integration to any target RCOS. The only dependencies are common base classes used by DFNs or DFPCs and the types that are used as inputs and outputs. This also simplifies the integration in the log replay tool of CDFF-Dev.

Testing DFNs or DFPCs offline with log data is possible with the provided Python bindings. Logs are replayed with a data flow control module that emulates the communication layer of an RCOS and a log player that replays logged data chronologically. Two essential elements are needed for a user to be able to replay data logs from a desired RCOS: (1) a conversion from the data log format used by the RCOS to an intermediate format that is used by InFuse, and (2) a data-type conversion from the RCOS to InFuse data types.

MessagePack[1] is the intermediate log file format that can be handled by CDFF-Dev. An example of the intermediate log format is shown in Fig. 4. A converter from ROCK's log format pocolog is already available, and a ROCK base-types to InFuse data types is under development. The converters will be stored in open repositories to ease its reuse between developers. The intermediate log format can be loaded as InFuse types (C++) wrapped in Python. These will be given as input to the Python interface of DFNs or DFPCs.

To replay log files, the user would provide the path to the logged data. While replaying log data, an orchestrator can suggest which DFPCs should be activated or deactivated. It will analyze incoming log data and the output of each active DFPC. The orchestrator in the final deployed setting receives requests from the Autonomy Framework (see Fig. 1, 2). When testing on CDFF-Dev the user, or a script, generates this requests. The Orchestrator then activates the DFPCs that produce the requested data products or triggers the operations that generate them. Additionally, the orchestrator can store the data in the Central Data Product Manager for later access.

The current state of the system is stored in an EnviRe graph [9] while replaying log data. EnviRe provides various tools to store and handle environment representations.[2] This data structure can be displayed with the EnviRe visualiser. Objects for which a visualisation has been implemented can be displayed with the EnviRe visualiser, for example, point clouds, laser scans, or poses.

For the development, analysis and comparison of new DFNs that use signal processing or machine learning algorithms, the framework pySPACE [10] is integrated in CDFF-dev.[3] Log files can be annotated while replaying log files and converted to a format that can be used by pySPACE. pySPACE provides numerous algorithms for signal processing and machine learning.

[1] https://msgpack.org
[2] http://envire.github.io
[3] https://pyspace.github.io/pyspace

They can easily compared with respect to various performance metric. The comparison can be run in parallel on a cluster.

```
{'/component.port':
    [{# sample 1
      'sourceFrame': 'A',
      'targetFrame': 'B',
      'timestamp':
          {'microseconds':0},
      'pos': [0, 0, 0],
      'cov_position': [ ... ],
            ...
     },
     {# sample 2 ... }, ... ],
 '/component.port.meta':
    [
     'timestamps': [ 1, 2, ... ],
     'type': 'RigidBodyState'
    ],
    ...
}
```

*Figure 4. Example of the intermediate log format in Python syntax*

### 2.4. Integration of data fusion solution in RCOSes

As part of our will to address a large community of users and ease the technological transfer from R&D studies to space products, the CDFF offers a convenient way to integrate DFNs/DFPCs into major open source robotics middlewares, namely ROS, ROCK, GenoM3 [11] and YARP [12]. The core of the proposed approach is to use ASN.1 to specify data structures and binary serialisation mechanisms (implemented by ASN1SCC, an open source ASN.1 compiler for embedded systems [13]), in order to exchange data between components and also across the RCOS.
To insure compatibility between the RCOS and programming languages like C++, python, javascript, and others, a common and basic ROS message (`asn1_bitstream.msg`) has been defined to transport the serialised ASN.1 data structures (see Fig. 5). Almost all RCOSes provide some compatibility with ROS messages and communication protocol, and should be able to use the proposed message. The message is structured as follow:

```
std_msgs/Header header
# Message type
string type
# Serialisation method : 0 (UPER)
uint8 sermethod
# Serialised data
uint8[] buf
```

where the field `buf` contains serialised data.
The Unaligned Packed Encoding Rules (UPER) are used by default for optimal compactness and encoding efficiency with low memory and low CPU footprints. However, Basic Encoding Rules (BER) or XML Encoding Rules (XER) could be used to facilitate communications with different programming languages. For instance, we are using BER to communicate with web applications through YARP.

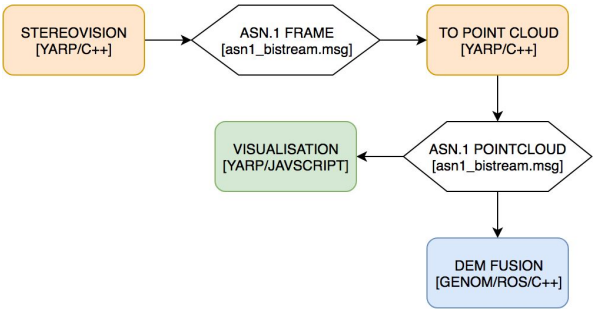



*Figure 5. CDFF integration principle in an heterogeneous environment of RCOS. A simple ROS message is defined to transport serialised ASN.1 data structures.*

This has been applied to GenoM3 with ROS and YARP, and shown all the expected benefits. Moreover, the use of YARP allows us to go even further towards an optimal integration, as its allows local communication; in that particular case ASN.1 data structures are exchanged without any serialisation.
There are many important advantages to the proposed method: clear data type / interface management with ASN.1, smooth integration effort in RCOS with ASN.1, enable inter RCOS communications, favor the separation between the RCOS and algorithms (this is even more true when GenoM3 is used).

## 3. DATA FUSION METHODS

The primary applications of InFuse are localisation and mapping, environment reconstruction, and object detection and tracking both in orbit and on the surface of other planets. We are currently implementing a number of DFPCs that address these applications, and will be released together with the framework itself, for others to use freely. The current list is given in Tab. 2.
Detailed descriptions of all those DFPCs is provided in [D5.5] and [D5.6]. Short descriptions of some of them are given below as examples:

- **3D Environment Reconstruction**: images collected by a stereo or a mono camera are projected in 3D coordinates and merged together by the estimation of an appropriate transform.
- **Short and Medium Range Object Detection**: 3D features are extracted from the 3D environment

map and a 3D model of the object, their matching identifies the object and its position.

- **Short and Medium Range Object Tracking**: a Kalman filter improves the estimated pose computed by the object detection DFPC.

| DFPCs for use in orbital situations |
|---|
| Far-Range Object Tracking<br>Mid- and Close-Range Target Detection<br>Mid- and Close-Range Target Tracking<br>Lidar-Based Tracking of a Target<br>Mid- and Close-Range Visual Tracking of a Target<br>3D Reconstruction<br>3D Tracking |
| **DFPCs for use in planetary situations** |
| Visual Odometry (LAAS-CNRS)<br>Visual Odometry (Magellium)<br>Absolute Localisation<br>Digital Elevation Mapping<br>Lidar-Based SLAM<br>Lidar-Based Localisation<br>Visual SLAM<br>Visual Map-Based Localisation<br>Far-Range Tracking<br>Mid-Range 3D Model Detection<br>Mid-Range 3D Model Tracking<br>Point-Cloud Model-Based Localisation with ICP<br>Point-Cloud Matching<br>Image Feature Detection and Matching<br>Point-Cloud Triangulation and Construction<br>Point-Cloud Model-Based Localisation with SHOT-Based Matching<br>Bundle Adjustment and Optimisation<br>Navigation Mapping<br>Position Manager |

*Table 2. Data Fusion Processing Compounds that are being implemented by the InFuse consortium as part of the Common Data Fusion Framework*

## 4. DEMONSTRATION SCENARIOS

The perception, localisation, and mapping capabilities of the CDFF will be evaluated in indoor test facilities and in outdoor planetary analog sites. Evaluations are planned in both orbital and planetary situations.

### 4.1. Orbital situations

For the orbital scenarios, the integrated software will be deployed on specialised test platforms for each rendezvous/on-orbit servicing scenario. It consists in three main DFPCs: detection, tracking, and reconstruction at various ranges. These DFPCs rely on the data obtained mainly from a camera, a LIDAR, IMU, or combination of them. In order to validate the performance, we simulate motion trajectories of various type for the target (linear, spinning, tumbling) under different space lighting conditions (nominal, under-illuminated, over-illuminated). These conditions depend on the direction of the sun with respect to the sensor main axis using an on-ground simulation facility [6], shown in Fig. 6.

*Figure 6. Ground test facility of DLR for visual target tracking when approaching a target in close-range approach and visual servoing*

The rendezvous and on-orbit servicing simulator consists of a mock-up including a servicer and a target satellite, mounted on a six degrees of freedom Kuka robots, and a sun simulator. A robotic arm is also mounted on the servicer satellite for on-orbit servicing tasks such as capturing the target and refueling. The arm incorporates stereo cameras that are used for the close-range DFPC. Fig. 7 shows what a camera image looks like during on-orbit servicing in space.

### 4.2. Planetary situations

The CDFF will provide state-of-the-art algorithms to implement necessary perception, localisation and navigation functions for planetary exploration rovers. Four functional use cases will sustain the development of algorithms in the project: long traverse localisation, long traverse navigation, rendez-vous and return to base. Each use case involves a limited set of key functions that will be evaluated.

The planetary scenario focuses on localisation and mapping within planetary environments. The sensors used include stereo vision, LIDAR, and inertial measurement. Navigation over long distances is enabled with DFPCs for localisation and for production of a Digital Elevation Map over long distances (~1km),

guidance to a defined objective point on the map and rendezvous with a target there, and return-to-base functionality once the above objectives are met. In addition, DFPCs provide the capability to build 3D point cloud environment models incrementally through structure-from-motion and SLAM methods.

Demonstrations of planetary scenarios are planned at CNES (see Fig. 8) and DLR facilities (see Fig. 9) with accurate ground truth of the terrain and the robot, as well as on the desert of Morocco. These will involve 5 different rovers from three different institutions, CNRS/LAAS, DLR and DFKI (see Fig. 9, 10 and 11).

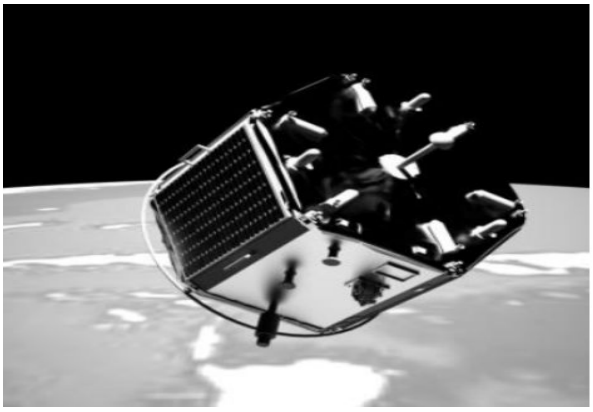


*Figure 7. A camera image representative of an on-orbit servicing scenario, with the Earth and deep space in background to the target.*

*Figure 8. CNES/SEROM test field*

*Figure 9. BB2 rover at DLR facilities for evaluation*

*Figure 10. CNRS/LAAS Mana and Minnie rovers*

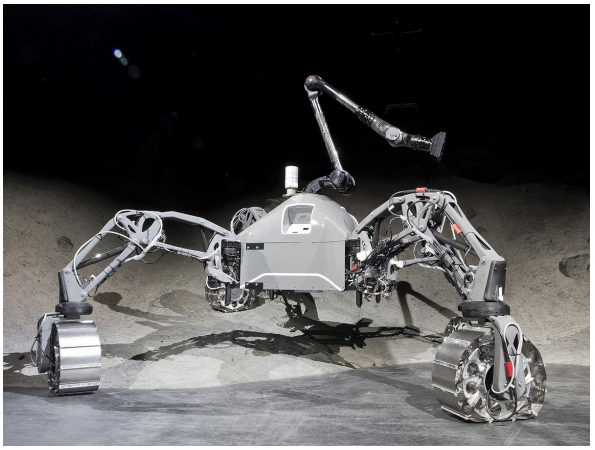


*Figure 11. Sherpa Rover from DFKI*

## 5. CONCLUSION

The Common Data Fusion Framework (CDFF) environment for development, testing and deployment of perception, localization and mapping algorithms in space robotic systems has been presented. The framework architecture has been designed to produce solutions with highly reusable components: *Data Fusion Nodes*, *Data Fusion Processing Compounds*, *Orchestrator* and *Data Product Manager*. Furthermore, it allows to describe, implement and test offline the software independently of the Robotic Control Operating System that the final robotic system might use.

A brief overview of the data fusion algorithms included in the CDFF has been provided, as well as how these are categorised and described in the framework through their description templates. Finally, the demonstration scenarios, which involves an orbital and a planetary track, have been briefly described.

ACKNOWLEDGEMENTS

The InFuse project is funded under the European Commission's Horizon 2020 Space Strategic Research Cluster, Operational Grants, grant number 730014.

REFERENCES

[1] Ocón J. et al., (2017) The Space Automation and Robotics General Controller (SARGON). In Proc. ASTRA 2017.

[2] Muñoz M. et al., (2017) ESROCOS: A Robotic Operating System for Space and Terrestrial Applications. In Proc. ASTRA 2017.

[3] Ocón J. et. al., Autonomous Controllers and Frameworks for Space Missions: GOTCHA and ERGO. In Proc. ASTRA 2017 .

[4] Govindaraj S. et al., (2017) InFuse: A Comprehensive Framework for Data Fusion in Space Robotics, ASTRA 2017.

[5] I3DS design documents - http://i3ds-h2020.eu/publications/deliverables

[6] Perrotin M. et al., (2010) The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software. In Proc. Embedded Real Time Software and Systems 2010, Toulouse.

[7] Oumer N.W. et al. (2018). Orbital Reference Implementation and Associated EGSE: Detailed Design. Deliverable 5.1 of the Horizon 2020 InFuse Project. URL: https://www.h2020-infuse.eu/documents.

[8] Souvannavong F. et al. (2018). Planetary Reference Implementation and Associated EGSE: Detailed Design. Deliverable 5.2 of the Horizon 2020 InFuse Project. URL: https://www.h2020-infuse.eu/documents.

[9] Hidalgo Carrió J., Arnold S., Böckmann A., Born A., Domínguez R., Hennes D., Hertzberg C., Machowinski J., Schwendner J., Yoo Y.H., & Kirchner F. (2016). EnviRe: Environment Representation for Long-term Autonomy. In Proc. Workshop on Artificial Intelligence for Long-Term Autonomy, International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16-20 May 2016. URL: https://sites.google.com/site/icra2016ailta.

[10] Krell M.M., Straube S., Seeland A., Wöhrle H., Teiwes J., Metzen J.H., Kirchner E.A., & Kirchner F. (2013). pySPACE: A Signal Processing and Classification Environment in Python. Frontiers in Neuroinformatics 7(40). PMID: 24399965. DOI: 10.3389/fninf.2013.00040. Google Scholar: 2961988771639493149.

[11] Mallet A., Pasteur C., Herrb M., Lemaignan S., & Ingrand F. (2010). GenoM3: Building Middleware-Independent Robotic Components. In Proc. International Conference on Robotics and Automation (ICRA), Anchorage, Alaska, United States, 3-7 May 2010. DOI: 10.1109/ROBOT.2010.5509539. Google Scholar: 9501243468305366893.

[12] Paikan A., Pattacini U., Domenichelli D., Randazzo M., Metta G., & Natale L. (2015). A Best-Effort Approach for Run-Time Channel Prioritization in Real-Time Robotic Application. In Proc. Intelligent Robots and Systems (IROS), Hamburg, Germany, 28 September - 2 October 2015. DOI: 10.1109/IROS.2015.7353611. Google Scholar: 12367842943333947581.

[13] Mamais G., Tsiodras T., Lesens D., Perrotin M. (2011). An ASN.1 Compiler for Embedded/Space Systems. In Proc. Embedded Real Time Software and Systems (ERTS2), Toulouse, France, 1-3 February 2012 Google Scholar: 13449943365769046250.