Deliverable Reference	:	D12.6
Title	:	Common Data Fusion Framework Final Package
Confidentiality Level	:	PU
Lead Partner	:	DFKI
Abstract	:	This document presents the software packages of the Common Data Fusion Framework.
EC Grant N°	:	730014
Project Officer EC	:	Christos Ampatzis (REA)

InFuse is co-funded by the Horizon 2020
Framework Programme of the European Union

DOCUMENT APPROVAL SHEET			
	Name	Organization	Date
Prepared and cross-reviewed by:	Alexander Fabisch Raul Dominguez Romain Michalec Shashank Govindaraj Irene Sanz Andrea De Maio Vincent Bissonnette Nassir Oumer	DFKI Strathclyde SPACEAPPS LAAS MAG DLR	15/02/2019

DOCUMENT CHANGE RECORD				
Version	Date	Author	Changed Sections or Pages	Reason for Change / RID No
0.1	08/01/2018	Alexander Fabisch	All	Basic document template
0.2	30/01/2018 04/02/2018	Romain Michalec	All	Add content and improve formatting
0.3	11/02/2019	Shashank Govindaraj	Section 4 Section 5	Requests for inputs from specific partners
1.0	15/02/2019	Shashank Govindaraj	All	Final review
1.1	29/03/2019	Irene Sanz	All	Modifications to address RIDs and update links

Executive Summary

This document describes the Common Data Fusion Framework (CDFF), that is to say the main software product of the InFuse project. As explained in previous deliverables, the framework is made up of three software components: CDFF-Core, CDFF-Support, and CDFF-Dev. This deliverable describes the state of those three components, and where to get them and their documentation.

Table of Contents

1 Introduction

1.1 Purpose

This deliverable documents the Common Data Fusion Framework (CDFF), that is to say the main software product of the InFuse project, written in C++ for the Core and Support components, and in Python for the Dev component. It explains where the components are downloadable from, where are their installation and usage instructions, how to get the optional proprietary dependencies of the Core component, which DFNs and DFPCs are in which branch, which support tools are available in the Support component, and which development tools are available in the Dev component.

1.2 Structure

This document is structured as follows:

Section 1 Introduction
Section 2 Description of the public release
Section 3 Optional proprietary dependencies of the CDFF
Section 4 DFNs
Section 5 DFPCs
Section 6 Support tools
Section 7 Development tools
Section 8 Conclusion

1.3 Applicable documents

AD1 InFuse Grant Agreement
AD2 InFuse Consortium Agreement
AD3 InFuse internal management manual for project partners

1.4 Reference documents

RD1 Description of Action document
RD2 D4.2: Advanced CDFF Architecture and ICD
RD3 D9.2: Data products management software
RD4 D9.4: Middleware and facilitators software
RD5 D11.4: TRR Ready CDFF

1.5 Acronyms

DEM: Digital Elevation Map
DFN: Data Fusion Node
DFPC: Data Fusion Processing Compound

EGSE: Electrical Ground Support Equipment
RI: Reference Implementation

2 Public release

The "public release" of the CDFF is made up of all the CDFF-related source code that can be made available to the community under an open-source license: the CDFF itself, and a couple open-source dependencies hosted for convenience in the same code repository as the CDFF, instead of in their own.

The expression "public release" stands in opposition to "SRC release", documented in the next section, which only means that the public release has been complemented with a few proprietary dependencies. There is no difference in the source code of the CDFF itself, or in the couple co-hosted open-source dependencies. The difference is only in the availability of those additional proprietary dependencies. See the next section for more information on the optional proprietary dependencies.

	CDFF code repository	**CDFF_dev code repository**
Components	CDFF-Core and CDFF-Support	CDFF-Dev
License	2-clause BSD	GPL v3 or later
Language	C++, CMake, Shell	Python
URL	https://gitlab.com/h2020src/og3/cdff	https://gitlab.com/h2020src/og3/cdff_dev
Documentation and installation instructions	Dependencies: CDFF/External/README.md CDFF: CDFF/README.md#building-the-cdff-core-and-support	Dependencies of CDFF-Dev: CDFF_dev/README.md#dependencies-of-cdff-dev CDFF-Dev: CDFF_dev/README.md#compiling-and-installing-cdff-dev Installation of CDFF-Core/Support and CDFF-Dev with Autoproj: cdff-buildconf/README.md#infuse-framework-install-instructions (branch: cdff_dev)
Additional documentation	How to write a DFN: https://docs.google.com/document/d/1hFTRKgJNN3n_brT3aajMA03AR_jQ2eCo-ZM33ggY5cE/edit?usp=sharing How to write a DFPC: https://docs.google.com/document/d/1ZUhZPnedd1mO42y-q4N7USltOnKeZzbyyZz_yzpLsmk/edit?usp=sharing	Software design of CDFF_Dev: https://docs.google.com/document/d/1yz_w7Eut6Rtg0d4I6R4mze2G8Oip4agyqrTDlKVgC6g/edit?usp=sharing

	Software design of CDFF-Support: https://docs.google.com/document/d/1BzKnNrRw6yIFIlrITiEGZXD8awtsmvNslqRuB4j29mw/edit#heading=h.8wiledc8qcdc	

In addition to the CDFF itself, we are also releasing a Docker image that contains an Ubuntu 16.04 LTS distribution and all the open-source dependencies of CDFF-Core, CDFF-Support, and CDFF-Dev, in the versions that we have used for development and testing. This image therefore makes up a reference, containerized environment for the CDFF, where one can build CDFF-Core/CDFF-Support and test it, or use the development tools provided by CDFF-Dev. It is called `h2020infuse/cdff` and is available publicly in the Docker Hub registry: https://hub.docker.com/u/h2020infuse.

We have also written extensive documentation about what Docker is, why and how to use it, how to startup containers from the InFuse Docker image, and how to mount local clones of the CDFF and CDFF-Dev repositories inside those containers. This documentation is available at the following address: https://drive.google.com/open?id=1aW3_giavOZdvOljEEfun4W0Cq2tlnDvb8S3y2bysjpw.

2.1 Files in the CDFF repository

Files in the CDFF repository are organized in a rather straightforward and self-explanatory manner. The following tree structure documents what each subdirectory is for. The CMake files of the build system are omitted for clarity.

```
External/
# The dependencies of the CDFF can be installed using the system's package manager,
# by compiling them from source, or a combination of both, at the user's
discretion.
# They are also shipped in a public Docker image available as h2020/infuse:latest
# from Docker Hub. In this directory, we provide tools for automating the
installation
# of most dependencies from source, to save everyone the trouble of retrieving each
# external library independently.
      installers/
      # Shell scripts used to install the dependencies of the CDFF from source
      patches/
      # Patch files for the dependencies that we need to patch before compiling
      get-cdff-dependencies.sh
      # Main dependency installer script. The user can set an installation prefix
      # with the -i option; to install (for instance) in
      External/install/{include,
      # lib} instead of /usr/local/{include,lib}, in case the latter contain
      # libraries of the same name, built with different options for other
      projects.
```

```
        README.md
        # Documentation about the dependencies of the CDFF and how to install them
        .gitignore
        # Contains the patterns: /sources/, /install/, /packages/

Common/
# Software components used by all or some of the DFNs and DFPCs
        Types/
        # Data types used throughout the CDFF
                ASN.1/
                        ESROCOS/
                        # ASN.1 data types adapted from (an early 2018 version of)
                        # ESROCOS
                                Frame.asn
                                Pointcloud.asn
                                Sonar.asn
                                …
                        InFuse/
                        # ASN.1 data types written by us
                                VisualPointFeatureVector2D.asn
                                CorrespondenceMap2D.asn
                                …
                C/
                # C data types transcompiled from their ASN.1 definition
                        Frame.{h,c}      # generated from ASN.1
                        Pointcloud.{h,c} # generated from ASN.1
                        Sonar.{h,c}      # generated from ASN.1
                        …
                CPP/
                # C++ wrappers encapsulating some of the C data types in C++ classes,
                # for convenience
                        Frame.{hpp,cpp}
                        Pointcloud.{hpp,cpp}
                        …
                README.md
                # Instructions on how to transcompile the ASN.1 types to C, or how to
                # download transcompiled types from the continuous integration server
                .gitignore
                # Contains the pattern: /C/
        Converters/
        # Converters between ASN.1 types and library types, that is to say types
        used
        # inside a DFN, such as cv::Mat or pcl::Pointcloud
                FrameToMatConverter.{hpp,cpp}
                MatToFrameConverter.{hpp,cpp}
        Errors/
        # Assertion macros
        Helpers/
        # Handlers for configuration file parsing
```

```
        Loggers/
        # Handlers for warning and error messages
        Visualizers/
        # OpenCV and PCL based GUI viewers, for debugging and profiling purposes
        Core/
        # Most libraries that provide core functionalities for the various DFNs
        # of the CDFF (e.g. OpenCV, PCL…) are handled as external dependencies,
        # see the External/ directory. However some of those core libraries are
        # hosted here, especially when they don't have their own code repository.
        # They are all appropriately documented.
                README.md
                liborbslam
                …
        …

DFNs/
# Each DFN is a set of .hpp/.cpp files in a adequately named directory:
        DepthFiltering/
        DisparityImage/
        FeatureExtraction2D/
        # Illustration of the structure of a DFN:
                FeatureExtraction2D_desc.yaml
                # YAML description of the FeatureExtraction2D DFN
                FeatureExtraction2DInterface.{hpp,cpp}
                # Interface and implementation of the DFN Interface
                # of the FeatureExtraction2D DFN
                HarrisDetector2D.{hpp,cpp}
                # Interface and implementation of the Harris-based DFN Implementation
                # of the FeatureExtraction2D DFN
                OrbDetectorDescriptor.{hpp,cpp}
                # Interface and implementation of the ORB-based DFN Implementation
                # of the FeatureExtraction2D DFN
        FeatureExtraction3D/
        FeatureDescription2D/
        FeatureDescription3D/
        FeatureMatching2D/
        FeatureMatching3D/
        ImageFiltering/
        ImageRectification/
        LidarBasedTracking/
        PrimitiveFinder/
        StereoRectification/
        StereoSlam/
        …
        DFNCommonInterface.hpp
        # This file is the DFNCI: the parent class of all DFNs and therefore the
        # grandparent class of all DFN Implementations

DFPCs/
```

```
# As with DFNs, each DFPC is a set of .hpp/.cpp files in a adequately named
directory:
        AbsoluteLocalization/
        HapticScanning/
        LIDARPoseGraphSlam/
        ModelBasedTracker/
        Reconstruction3D/
        VisualSlamStereo/
        …
        DFPCCommonInterface.hpp
        # This file is the DFPCCI: the parent class of all DFPCs and therefore the
        # grandparent class of all DFPC Implementations

Documentation/
# Directory where the Doxygen documentation can be generated

Support/
# Two components of CDFF-Support (the third one, the DFPCs, have their own
directory):
        CentralDPM/
        # The central data product manager
        Orchestrator/
        # The orchestrator (WIP)

Tests/
# Various test executables of the CDFF. They are not, per se, a part of the CDFF.
They
# are built if the CMake option BUILD_TESTS is given at project configuration.
        UnitTests/
        # Unit tests of the CDFF. They are run by the continuous integration server
        # to ensure self-consistency and successful builds.
                Common/
                # Unit tests of the common libraries
                DFNs/
                # Unit tests of the DFNs
                DFPCs/
                # Unit tests of the DFPCs
                Support/
                # Unit tests of the CDFF-Support components
                # (except the DFPCs which have their own directory)
                Catch/
                # The Catch library, a library for unit testing
                README.md
                # How to write unit tests using the Catch library
        GuiTests/
        # Tests that display output using the OpenCV and PCL based viewers in
        Common/
        PerformanceTests/
        KeyPerformanceMeasuresTests/
```

```
        # Tests that output quantitative performance measures
        ConfigurationFiles/
        # Configuration files for the DFNs and the DFPCs used in the tests
        Data/
        # Sample data on which the tests run
        …

Tools/
# Tools for building, testing, debugging… Each tool in this directory is an
# independent helper application. If a tool consists of C++ code meant to be built
# with the rest of the project, it's more likely a part of the CDFF, and therefore
# meant to be elsewhere, Common/ perhaps.
        ASN.1/
        # The ASN.1 transcompiler that we are using is ESA's/ESROCOS's ASN1SCC, an
        # ASN.1 to C and ADA transcompiler which targets safety-critical embedded
        # systems and has been used in the TASTE project. More compilers in this
        list.
        # This directory contains scripts for transcompiling the ASN.1 types to C
        with
        # a downloaded-on-the-fly ASN1SCC compiler, and for downloading
        transcompiled
        # ASN.1 types from the continuous integration server
        CMake/
        # Uninstallation scripts for the CDFF. They enable "make uninstall" and
        # "make rm" commands.
        CPPCheck/
        # A static code analyzer used during continuous integration testing
        Valgrind/
        # A memory leak detection tool used during continuous integration testing
        Docker/
        # Dockerfiles to build the containerized environments in which the
        continuous
        # integration server, or a user, can build and test the CDFF. These Docker
        # environment contain an Ubuntu 16.04 LTS distribution completed with the
        # dependencies of the CDFF built from source and installed in /usr/local.

README.md
INSTALL.md
CONTRIBUTING.md
LICENSE.md
AUTHORS.txt
# Self-explanatory

.gitlab-ci.yml
# Configuration file for the continuous integration server. It defines the
pipelines
# that the server runs.

.gitignore
```

```
# Contains the patterns: *~, cmake_uninstall.cmake, cmake_purge.cmake, /build/
```

2.2 Files in the CDFF_dev repository

The following tree structure documents what each subdirectory in the CDFF_dev repository is for:

```
bin/
# Executables, mostly command line tools for code generation and log data handling
cdff_dev/
# Python library of CDFF-Dev
       dfpcs/
       # Python bindings of DFPCs
       dfns/
       # Python bindings of DFNs
       extensions/
       # Bindings for other code that is not a DFN or DFPC
       templates/
       # Jinja2 templates for code generators
       test/
       # Unit tests
       *.py
       # Core Python modules of CDFF-Dev
cpp_helpers/
# C/C++ helpers that will be used by the Python library
doc/
# Anything that is related to the documentation of CDFF-Dev
examples/
# Example scripts that demonstrate how CDFF-Dev can be used
test/
# Integration tests that combine two or more modules of CDFF-Dev
README.md
# Most important information about CDFF-Dev
requirements.txt
# Python dependencies of CDFF-Dev
setup.py
# Python setup script for CDFF-Dev
…
```

3 Optional proprietary dependencies of the CDFF

The CDFF itself does not contain any proprietary software, but a few of its software modules depend on proprietary libraries. Those modules are disabled by default, that is to say they will normally not be compiled. They can be enabled by anyone who has the proprietary libraries (and a license to use them).

The expression "SRC release", as opposed to "public release", means that in addition to the regular, public release, all or some of the proprietary dependencies are available to the user. It is important to note that "SRC release" does not mean a different CDFF, and so it is not a release at all, and the expression is a bit unfortunate: it is the same source code, released on the same website. The difference is only in the availability of the proprietary dependencies.

3.1 EDRES

EDRES is a proprietary robotics library developed and owned by the Centre National d'Études Spatiales in France (CNES). It provides a wide range of functionalities centered around perception for planetary robotics, such as accurate and robust stereo correlation, navigation map building and path planning.

3.1.1 Additional features

As agreed among partners, most of the functionalities provided by the EDRES components of the CDFF are also present in the open-source release. The functions provided by EDRES are, however, considered to be more accurate, faster or less resource intensive, depending on the case. One exception to this open-source/proprietary function duality is the navigation map building functionality, which is only provided with the EDRES library, as it was an extra functionality useful for representative test campaigns, but not within the scope of InFuse.

3.1.2 Installation instructions

To obtain the EDRES library, the interested user must formally request it by contacting the coordinator of the InFuse consortium, or Magellium, the partner responsible for EDRES functionalities. Contacts: thierry.germa@magellium.fr or vincent.bissonnette@magellium.fr.

A software archive containing the EDRES library in binary form, its header files, and a support file for CMake, will be sent to the user. Installation can be performed with the CDFF-provided command:

```
External/get-cdff-dependencies.sh -d edres-wrapper
```

See the documentation in `External/installers/edres-wrapper.sh` to know where to put the software archive you got from Magellium before executing this command, and see the usage message from `External/get-cdff-dependencies.sh --help` for information on the available options, such as the installation prefix.

After the library is installed, the CDFF can be built (or rebuilt) with EDRES-dependent DFNs by setting the following CMake cache entry (for instance with a `-D` option on the command line): `HAVE_EDRES=ON`.

3.1.3 Conditions of use

The complete terms of use of the EDRES library will be provided with it to every user of the EDRES library.

3.2 DLRTracker-core

DLRTracker-core is a proprietary C++ library that provides functions to support visual tracking. It is developed by the Institute of Robotics and Mechatronics at DLR and targets embedded space systems.

3.2.1 Additional features

The following functionalities provided by the DLRTracker-core library have also been implemented as open-source DFNs for the public release of the CDFF: Kalman filtering, edge detection, and image gradient computation.

3.2.2 Installation instructions

To obtain the DLRTracker-core library, the interested user must formally request it to DLR's Institute of Robotics and Mechatronics, Department of Perception and Cognition.

A software archive containing the DLRTracker-core library in binary form, its header files, and a support file for CMake, will be sent to the user. Installation can be performed with the CDFF-provided command:

```
External/get-cdff-dependencies.sh -d dlrtracker-core
```

See the documentation in `External/installers/dlrtracker-core.sh` to know where to put the software archive you got from DLR before executing this command, and see the usage message from `External/get-cdff-dependencies.sh --help` for information on the available options, such as the installation prefix.

After the library is installed, run CMake in your build tree to have it detect the availability of the library and built (or rebuilt) the CDFF with DLRTracker-dependent DFNs.

3.2.3 Conditions of use

The DLRTracker-core library is proprietary software which can be used by DLR partners for non-commercial purposes within the framework of the PERASPERA strategic research cluster (SRC).

4 DFNs

4.1 Integrated in the CDFF

Previous deliverables have featured the list of the DFNs we aimed for. Here we list the DFNs that effectively exist in the `master` branch of the CDFF repository at public release time, in the `DFNs/` directory. Every DFN features a description file in YAML format, as planned in D4.2.

DFN	Description	Track
BundleAdjustment	Determines the poses of a camera from the images taken at those poses	OT and PT
CamerasTransform Estimation	Estimates the geometric transformation between two cameras based on pairs of 2D matching keypoints found in two images that the cameras captured	OT and PT
ColorConversion	DFN that convert an image from a colorspace to another (i.e. RGB to Grayscale or HSV to BGR)	-
DepthFiltering	DFN that filters a 2D depth image	-
DisparityImage	Creates a disparity image from an image pair	PT
DisparityToPointCloud	Creates a point cloud from a disparity image	OT and PT
DisparityToPointCloud WithIntensity	Creates a point cloud with intensities from a disparity image and a grayscale image	OT and PT
FeaturesDescription2D	Computes descriptors for 2D keypoints	OT and PT
FeaturesDescription3D	Computes descriptors for 3D keypoints	OT and PT
FeaturesExtraction2D	Extracts 2D keypoints from a 2D image	OT and PT
FeaturesExtraction3D	Extracts 3D keypoints from a 3D pointcloud	OT and PT
FeaturesMatching2D	Matches 2D keypoints	OT and PT
FeaturesMatching3D	Matches 3D keypoints	OT and PT
ForceMeshGenerator	Creates a point cloud using the feedback from a force sensor	OT and PT
FundamentalMatrix Computation	Estimates the fundamental matrix of a camera pair based on pairs	OT and PT
ImageDegradation	DFN that degrades the resolution of an image	PT

ImageFiltering	Applies an image processing filter to a 2D image	OT and PT
ImageRectification	DFN that rectifies an image	-
KFCorrection	Corrects or updates states with a given measurement in Kalman filtering	OT and PT
KFPrediction	Predicts current states given the last states in Kalman filtering	OT and PT
LidarBasedTracking	DFN that tracks a model in a point cloud	OT and PT
ModelBasedDetection	DFN that performs Linemod detection for object pose detection	OT and PT
PerspectiveNPoint Solving	Solves the Perspective-n-Point problem: estimates camera pose from 3D points and corresponding 2D points.	OT and PT
PointCloudAssembly	DFN that combines two point clouds together	OT and PT
PointCloudFiltering	DFN that applies a filter on a point cloud	OT and PT
PointCloudReconstruction 2DTo3D	Turns pairs of 2D matching keypoints into a reconstructed 3D pointcloud of keypoints	OT and PT
PointCloudTransformation	Transforms a point cloud	OT and PT
PoseEstimator	Estimates the pose of a robot given the primitives found in the image	OT and PT
PoseWeighting	Estimates the pose of a joint given different predictions	OT and PT
PrimitiveFinder	Finds primitives in a 2D image	OT and PT
PrimitiveMatching	DFN that finds primitives in a 2D image	-
Registration3D	Registers a source point cloud on a sink point cloud	OT and PT
StereoDegradation	DFN that degrades the resolution of an image	PT
StereoMotionEstimation	DFN that computes an estimated motion between two stereo acquisitions	OT and PT
StereoReconstruction	Turns a pair of stereo images into a reconstructed 3D scene (in the form of a 3D pointcloud)	OT and PT
StereoRectification	DFN that rectifies an stereo pair	PT and OT
StereoSlam	DFN that performs visual SLAM on a stereo image pair input. As this is a relative localisation technique, if tracking is successful, the pose output is expressed in	PT

	the reference frame of the first image passed to the tracker.	
Transform3DEstimation	Estimates the geometric transformation between two sets of matched 3d points	OT and PT
Voxelization	Voxelizes a 2D depth map	OT and PT

4.2 In a development branch of the CDFF repository

The following DFNs are works in progress and in other branches than `master`:

DFN	Description	Current location
OutlierDetection	Outlier detection node that uses machine learning https://gitlab.com/h2020src/og3/cdff/merge_requests/164	branch: outlier_detection

5 DFPCs

5.1 Integrated in the CDFF

Previous deliverables have featured the list of the DFPCs we aimed for. Here we list the DFPCs that effectively exist in the `master` branch of the CDFF repository at public release time, in the `DFPCs/` directory. Every DFPC features a description file in YAML format, as planned in D4.2.

DFPC	Description	Track
Model-Based Visual Tracking	Tracking of an object in 6 degrees of freedom given an image (or stereo images) and the object's 3D model. A wrapper around the DLRtracker_core library.	OT
Haptic Scanning	Reconstruction of the environment using a force sensor as input	OT and PT
Model-Based Tracking	Tracking of the pose of a robot in a scene given its physical description	OT and PT
Model-Based Point-Cloud Localization	DFPC that estimates the position of a given 3D point cloud (the model) in a larger 3D point cloud (the scene)	OT and PT
Reconstruction 3D	DFPC that builds a 3D point cloud model of a target from multiple stereo image pairs of it	OT and PT
Reconstruction And Identification	DFPC that builds a 3D point cloud scene from multiple stereo image pairs, and estimates the position of a given point cloud in the larger point cloud	OT and PT
Visual Stereo SLAM	Simultaneous localization and mapping based on stereo images. A wrapper around the StereoSlam DFN.	PT
Map-Based Visual Localization	Localization in a previously built SLAM map. This DFPC was not implemented as a separate DFPC: instead it is an alternate operating mode of the Visual Stereo SLAM DFPC.	PT

The following DFPCs have been implemented as DFNs instead, and those DFNs are in the `master` branch of the CDFF repository, in the `DFNs/` directory:

DFPC → DFN	Description	Track
Visual Odometry MAG	Visual odometry based on wheel odometry and stereo images and developed by Magellium	PT

Magellium's version of the Visual Odometry DFPC has been implemented as a single DFN instead, the StereoMotionEstimation DFN. This DFN depends on EDRES, a proprietary library by CNES. An open-source version of the Visual Odometry DFPC has been developed by LAAS, see Visual Odometry LAAS in the tables of this section.		
Model-Based Point-Cloud Tracking	DFPC that perform relative localisation wrt a known model described by its point cloud.	OT
This DFPC has been implemented as a DFN instead, the LidarBasedTracking DFN.		
Mid/Close-Range Model-based Detection	DFPC which detects a known object in stereo images from a pre-trained template and estimates a coarse relative pose.	OT
This DFPC has been implemented as a DFN instead, the ModelBasedDetection DFN.		

5.2 In a development branch of the CDFF repository

The following DFPCs are works in progress and are located in various development branches instead of the CDFF's `master` branch.

DFPC	Description	Track
DEM Building	Digital elevation mapping based on an input pointcloud (lidar-captured or reconstructed from stereo images) and the estimated pose of the rover	PT
This DFPC is in the LAAS_DFPCs branch: https://gitlab.com/h2020src/og3/cdff/tree/LAAS_DFPCs		
Visual Odometry LAAS	Visual odometry based on wheel odometry and stereo images and developed by LAAS	PT
This DFPC is in the LAAS_DFPCs branch: https://gitlab.com/h2020src/og3/cdff/tree/LAAS_DFPCs		
Navigation Map Building	Creation of navigation maps from dense DEM input.	PT
This DFPC has been implemented as a DFN instead, the DEMToNavMap DFN. This DFN depends on EDRES, a proprietary library by CNES. Because it doesn't have an open-source implementation, it has not been merged into the master branch. It can be found in the feature/yarp_middleware_support branch: https://gitlab.com/h2020src/og3/cdff/tree/feature/yarp_middleware_support/DFNs		
Mid/Close-Range Model-based Tracking	DFPC which performs relative localisation wrt to a known target specified by its CAD model using stereo images.	PT OT

This DFPC has been implemented as a DFN instead, the StereoModelBasedTracker DFN. This DFN can be found in the feature/model_based_tracking branch: https://gitlab.com/h2020src/og3/cdff/tree/feature/model_based_tracking A merge request has been issued and a first code review has been done.		
Pose Fusion [aka POM]	Integrate data corresponding to past poses, and produce updated past poses after application of a pose fusion process.	PT
https://gitlab.com/h2020src/og3/cdff/tree/LAAS_add_POMCore		

5.3 In a private repository

The following DFPCs are works in progress and are located outside the CDFF repository, in partners' servers.

DFPC	Description	Track
LIDAR Pose-Graph Slam	Simultaneously builds an environment model composed of a series of LIDAR point clouds, and provides pose estimates for the rover	PT
Absolute Localization	Absolute localization for a rover using orbital images, point clouds and orthophotos	PT

6 Support tools

6.1 Pose manager

The pose manager (POM) estimates the current pose of the robotic system from poses obtained from several sources.

It is in the branch `LAAS_add_POMCore` of the CDFF repository.

https://gitlab.com/h2020src/og3/cdff/tree/LAAS_add_POMCore

6.2 Central data product manager

The central data product manager (Central DPM) allows certain data types to be stored permanently on mass storage for later retrieval.

It is in the `master` branch of the CDFF repository, in the `Support/CentralDPM` directory.

6.3 Orchestrator

The orchestrator software maps the Autonomy Framework request to corresponding DFPCs and OG4 sensors operation mode (based on types of sensors). An orchestrator configuration file and sensor operation modes configuration are required for the specific system before deployment. The Orchestrator has to be extended with RCOS specific adapters to control the lifecycle of DFPCs based on OG2 requests and corresponding OG4 operation modes.

It is in the branch `feature/orchestrator` of the CDFF repository.

https://gitlab.com/h2020src/og3/cdff/tree/feature/orchestrator

6.4 ARM64/FPGA support

The branch `xlnx` of the CDFF repository contains the necessary instructions and the necessary configuration file (CMake toolchain file) to cross-compile a selected part of the CDFF - the 2D feature extraction DFN and the libraries in the `Common/` directory that this DFN requires - to the ARM64 microprocessor architecture (`aarch64`). It targets the Xilinx Zynq UltraScale+ ARM64/FPGA multiprocessor system-on-a-chip embedded system, reported on in detail in deliverables D10.3, D10.6, and D10.9.

This branch also contains the necessary instructions and the necessary configuration files (CMake toolchain file and Boost.Build project-config jamfile) to cross-compile to ARM64 all the necessary dependencies of this selected part of the CDFF.

Note that in order to proceed with cross-compilation, the user must provide an ARM64 sysroot and an ARM64 toolchain. We used those found in Xilinx's ReVISION software stack and in Xilinx's SDK, respectively.

Because the `master` branch targets the x86_64 architecture of regular desktop computers and the `xlnx` branch targets the ARM64 architecture of the Xilinx EGSE, the `xlnx` branch is not meant to be merged into the `master` branch.

https://gitlab.com/h2020src/og3/cdff/tree/xlnx

7 Development tools

7.1 DFN and DFPC code generators

The DFN code generator uses the description file of a DFN to write out C++ code for the interface of that DFN, and the corresponding Python bindings.

https://gitlab.com/h2020src/og3/cdff_dev/blob/master/bin/dfn_template_generator

The DFPC code generator does the same thing with a DFPC.

https://gitlab.com/h2020src/og3/cdff_dev/blob/master/bin/dfpc_template_generator

7.2 Data log replay

This software component replays log files to test data fusion solutions and visualize sensor data and fused data. It is made up of:

- DataFlowControl
  https://gitlab.com/h2020src/og3/cdff_dev/blob/master/cdff_dev/dataflowcontrol.py
- Log iterators
  https://gitlab.com/h2020src/og3/cdff_dev/blob/master/cdff_dev/logloader.py
- Replay
  https://gitlab.com/h2020src/og3/cdff_dev/blob/master/cdff_dev/replay.py

7.3 Middleware facilitation

7.3.1 Rock

7.3.1.1 pocolog2msgpack

Logfile converter from Rock's pocolog format (github.com/rock-core/tools-pocolog) to MessagePack (github.com/rock-core/tools-pocolog). An additional tool is used to convert types to the corresponding ASN.1 data types.

Location: https://github.com/rock-core/tools-pocolog2msgpack

7.3.1.2 rock2infuse

Converter to turn rock types stored in msgpack format into the CDFF's ASN.1 types in msgpack format which is the format that CDFF_Dev uses.

Location: https://github.com/rock-core/tools-pocolog2msgpack

7.3.1.3 ROS to ASN.1 and ASN.1 to ROS

Support for conversion of a set of standard ROS messages to ASN.1 types, and vice versa. Some of the ROS types are specific to robot systems that were used for data collection during internal tests.

Location: https://gitlab.com/h2020src/og3/cdff_ros

8 Conclusion

The state of the CDFF at public release time is in adequation with what can be expected from a team of researchers - who are not formally-trained software engineers but professional research experts - developing a large-scale, highly complex, state-of-the-art software product in a very limited time - slightly longer than a year for the actual implementation activities, plus a few months of architectural design. Some of the modules that we intended to develop require adaptations and improvements for completeness. A small subset of the Core and Support software are in development branches where they undergo corrections and polishing before they are deemed ready to be integrated into the main branch. However they have been implemented and tested, but require final reviews before being merged to the master branch.

Nonetheless, a lot of work has been done and has resulted in a unique sensor data fusion framework, which thanks to its modularity, its wide scope, and its common interfaces, will hopefully be helpful to the partners involved in the next round of SRC projects and to the robotics community at large. Thanks to the choice of open-source for both the CDFF and most of its dependencies, development and maintenance can continue openly on the internet, hopefully with the help of all interested parties worldwide.

Speaking of open-source, for the sake of the partners involved in the next OGs, it is important to stress again that the internal release of the CDFF for follow-up projects does not differ from the open-source release. It is the same source code in both cases. The difference comes from the availability, for the partners in the next round of OGs, of two proprietary dependencies which make it possible to compile a few DFN implementations and DFPC implementations that users of the open-source release cannot compile.