Title: Modularized PM_SLAM Internship Report

Creation Date: 20 September 2012

Prepared by: Piotr WECLEWSKI <weclewski.piotr@gmail.com>

Abstract:

I Document Change Record

Revision	Affected content	Description	Date	Initials
0.1	All	Template of document, content proposition	21.09.2012	PW
0.2	All	Detailed content list, graphics, implementation	25.09.2012	PW
0.3	Implementation	Detailed descriptions and diagrams of system implementation	26-27.09.2012	PW
1.0	All	First revision	27.09.2012	SA
1.1	All	Corrections after revision	28.09.2012	PW

II Contents

1 Introduction

1.1 SMART & PM_SLAM

– What is SMART,

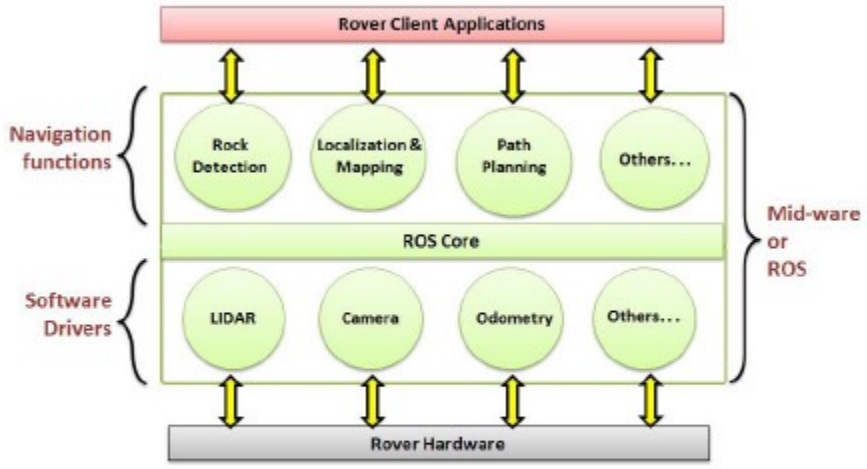
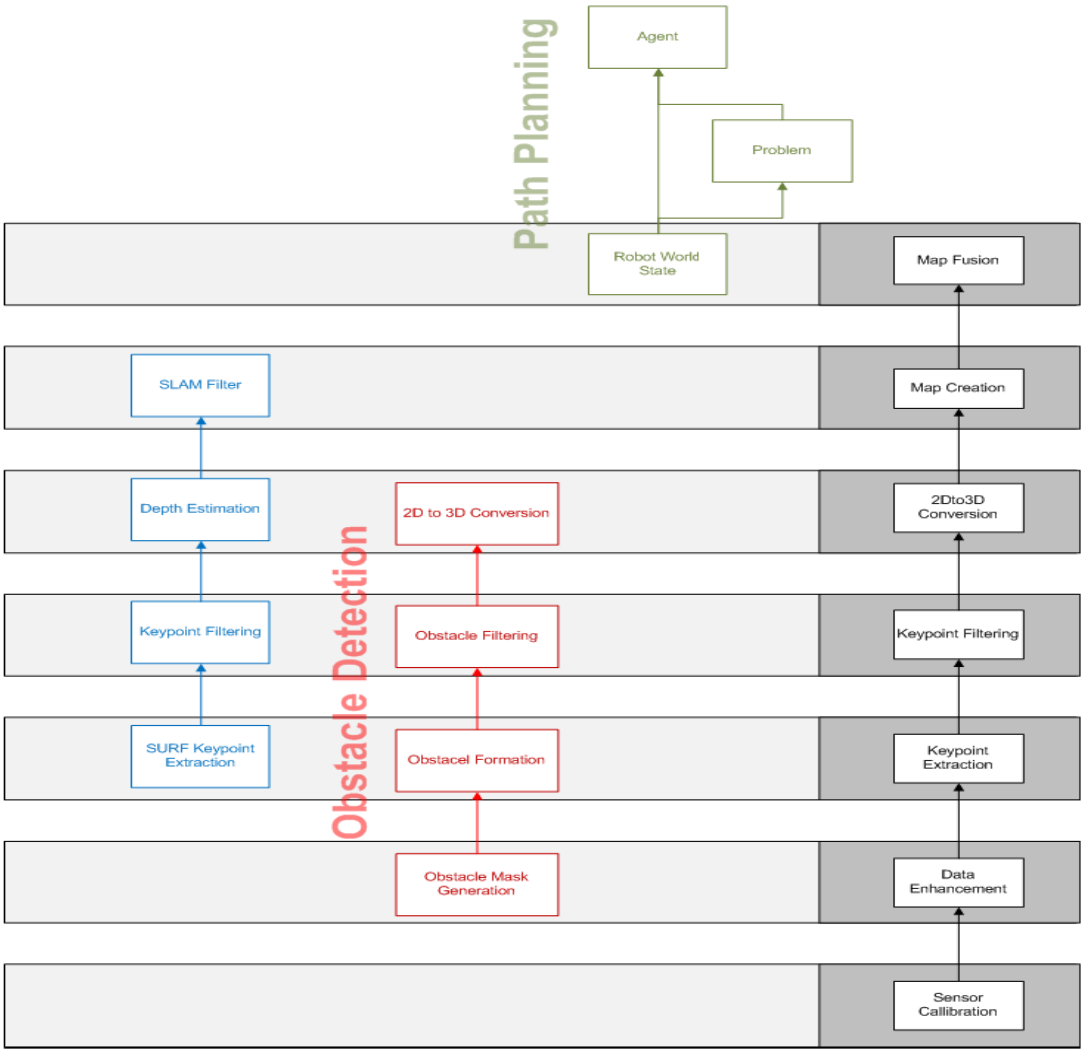
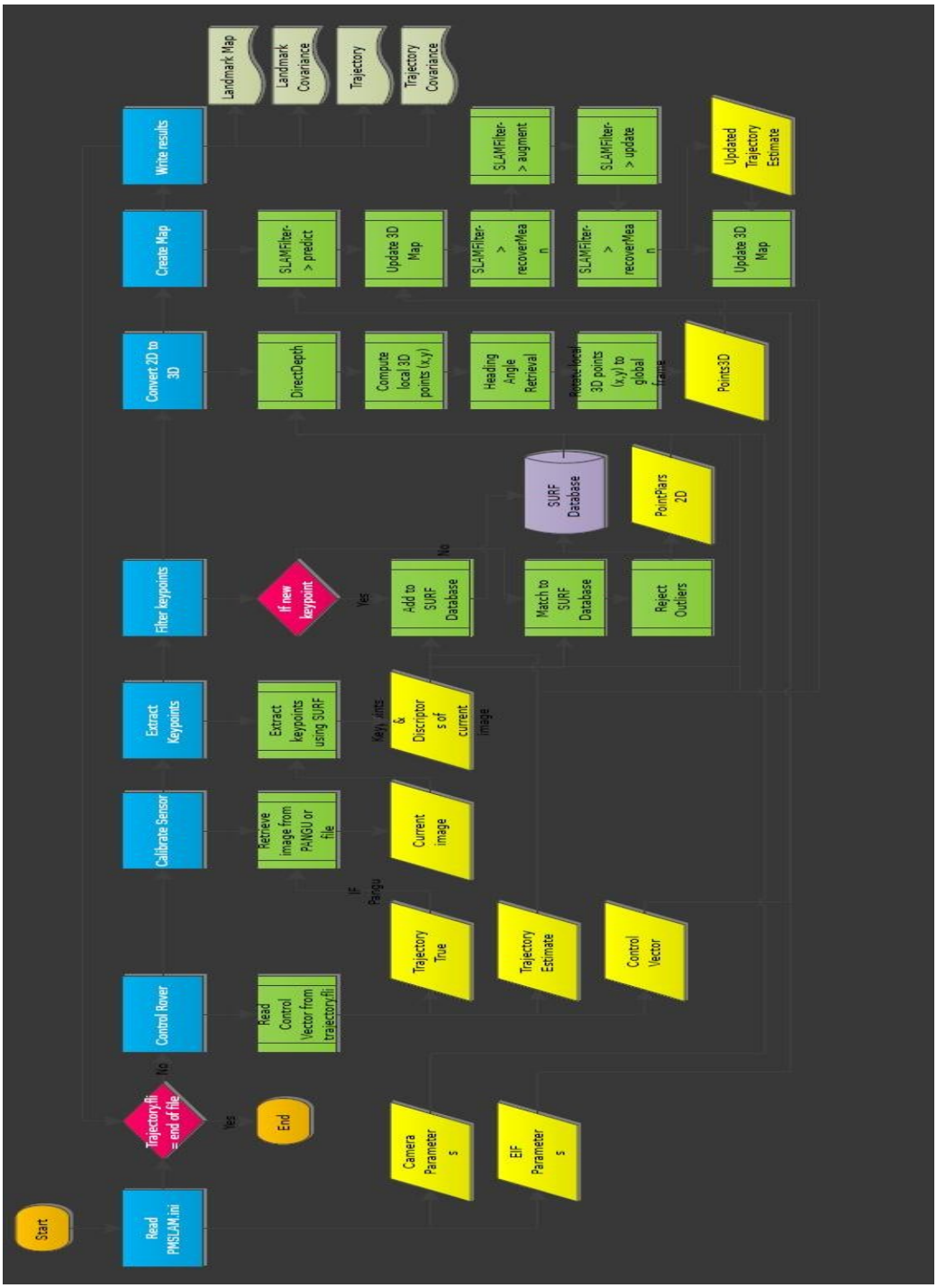


FIGURE 3. *Software system overview of SMART*

- Environment Modeling



– What is PMSLAM?



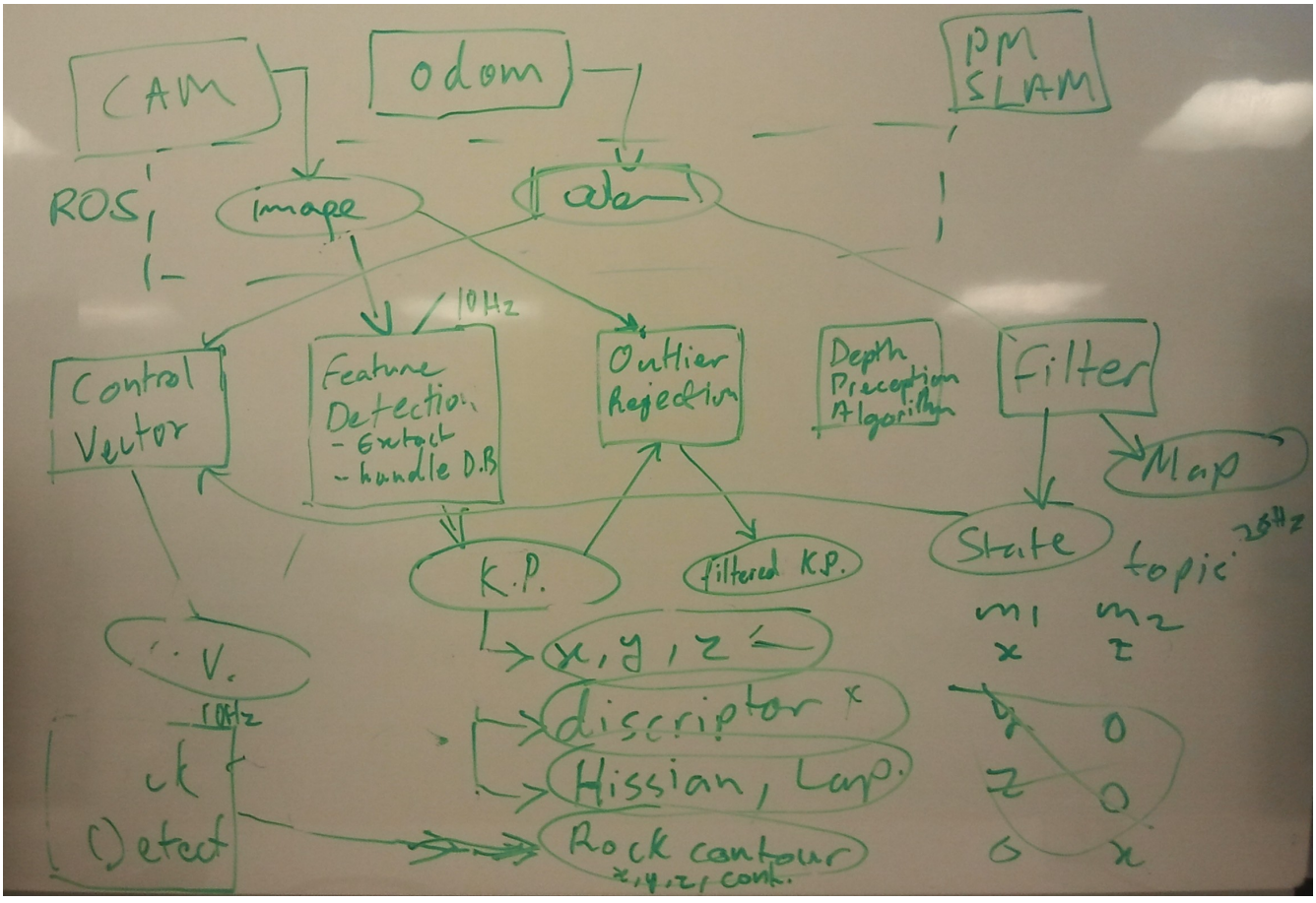
–

- introduction to functionality,
- modules roles and used techniques

1.2 Project Objectives

1.3 Literature Review

2 Preliminary Assumptions



3 Conceptual Design & Background Theory

3.1 Modularized Structure

- Modules with data exchanging
- Modularity – benefits and problems
- Advantages of modularizing PMSlam
- Challenges that modularization introduces

3.1.1 Input and output

Assumptions for MPMSlam input are the same as for PMSlam project and dictated by the data processing techniques. For monocular slam, input data are images from single camera. This project proposes to improve reliability and accuracy of dead reckoning navigation (navigation based only on odometry information) by monocular slam technique. Odometry information is also used for verification of monocular navigation output.

The main output form MPMSlam are a 3D map based on tracked, robust features and corrected position information. Additionally there are sets of data that are used for testing the performance and accuracy estimation.

3.1.2 Modules Role

3.1.3 Data Flow

[diagrams from resturing PMSLAM]

3.2 ROS

- (ros distributed structure pic./dia.)
- benefits of using ROS framework
  - common, standardized data structures, units, etc.
  - data exchange handling via ethernet network
  - distributed processing
  - universal tools – work analysis, debugging, visualization, etc.
- disadvantages

- non-real time system
-

3.3 Separation of ROS Interface and Processing Methods

Separation of ROS interface and PMSlam methods involves independence of PMSlam code on ROS data structures and functions. It is important because PMSlam methods could be used in projects where ROS system is not recommended or not available. Obviously, in applications requiring high performance, pure C++ code will execute faster than the one based on ROS system. In this kind of applications there is simple posibility to integrate PMSlam methods code into pure C++ program without ROS system load. Picture below presents core parts of separation concept and corelation between independent parts of package.

Project package

ROS interface	PMSlam functionality
Handling ROS and system events, requests and responses	Implementation of processing algorithms for research
Use of init(), process() methods	Implementation of init(), process() methods with concreate algorithms
Work with standard/auto-generated (publishing, subscribing, conversions) and pass it through methods parameters.	Work with custom data representation
Feature abstract-interface operations	Operations on specialized objects inherited from abstract interface

3.4 Features Representation

Feature representation is a key part of whole MPMSlam project. The concept of generic feature representation includes an abstract-interface and a concreate definition of feature and feature set structures. Final declaration of this structures inherits abstract class, which forces overwrite and implementation of interface methods. Abstract interface is used in places, where data processing should remain independent of concreate features representation such as features exchange between modules, matching in database or storing to log files. Detailed implementation is described in chapter Implementation.

3.5 Custom Data Exchange

Because feature representation is based on non-ROS messages structures, that require to implement custom way to pass feature data through ROS topics. For this reason feature structures has methods to serialize and deserialize data.

Serialization is an operation of translating data from complex structure to linear, byte order buffer which allows incapsulating this data into a ROS message and publish it. In other part of the system, the message is received and deserialized again to a specific data structure.

3.6 Database Handling (in progress)

With an implemented generic feature representation, it is possible to move database operations, such as searching in database or matching feature with database content, into a separated package. MPMSlam project requires initialization of database with first set of features, adding new features o database, removing chosen record, matching feature (or set of features) with database content and getting chosen and all records from storage. When database becomes larger and larger, getting whole content could be harmful for communication performance, nevertheless only PMSlam Filter requires this operation for building the map based on feature positions. Due to PMSlam Filter being the slowest part of the project, it is possible to perform this operation on request (as a service) instead of publishing the whole database in a continuos loop.

4 Implementation

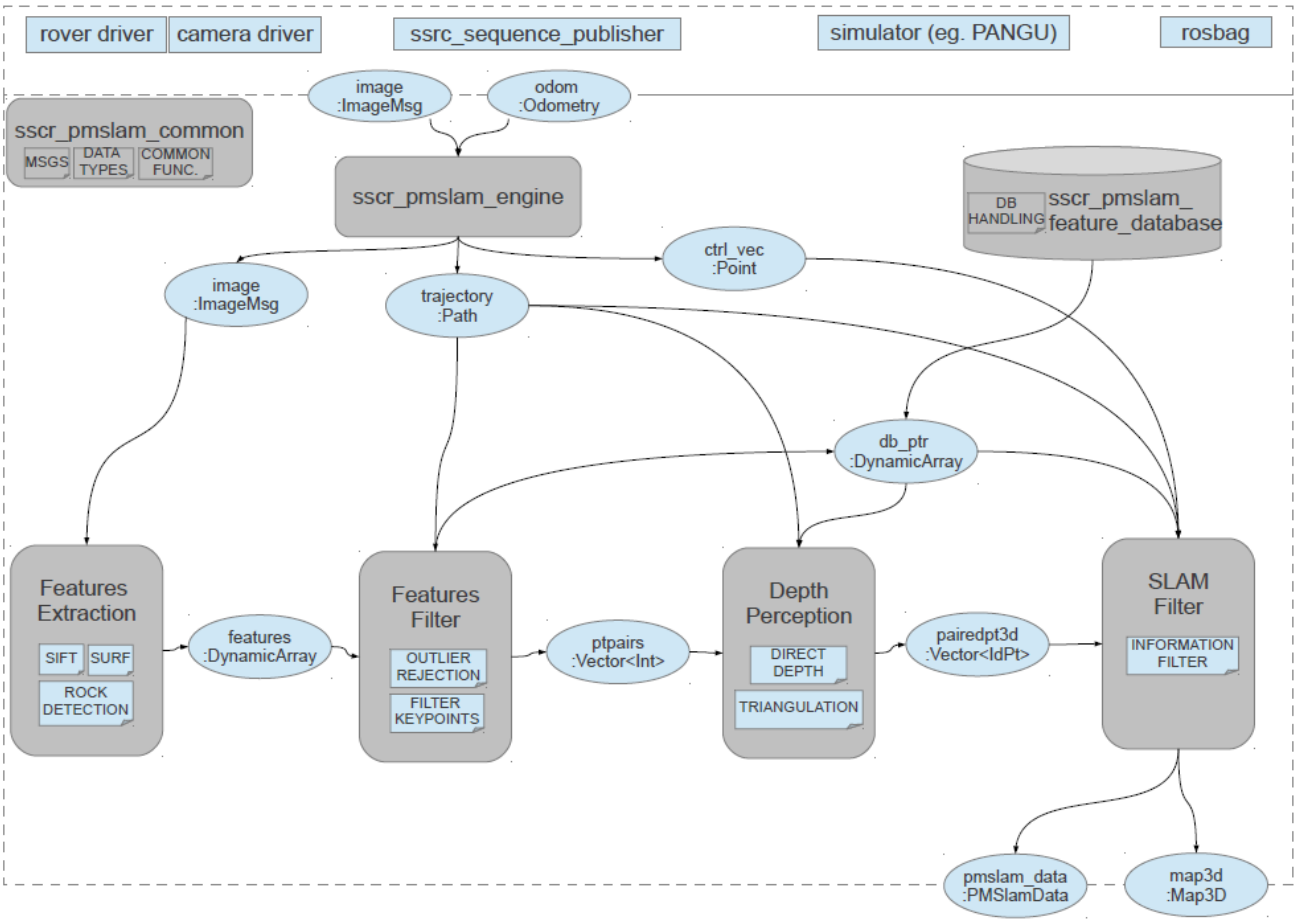
Each class in the whole system is defined (in file *.h) and declared (in file *.cpp) in own files pair. File and class coded in it always have the same name. MPMSlam project was implemented following ROS C++ Style Guide (http://www.ros.org/wiki/CppStyleGuide), which forcing the same naming convention and structure for all code.

4.1 Overall Unifications

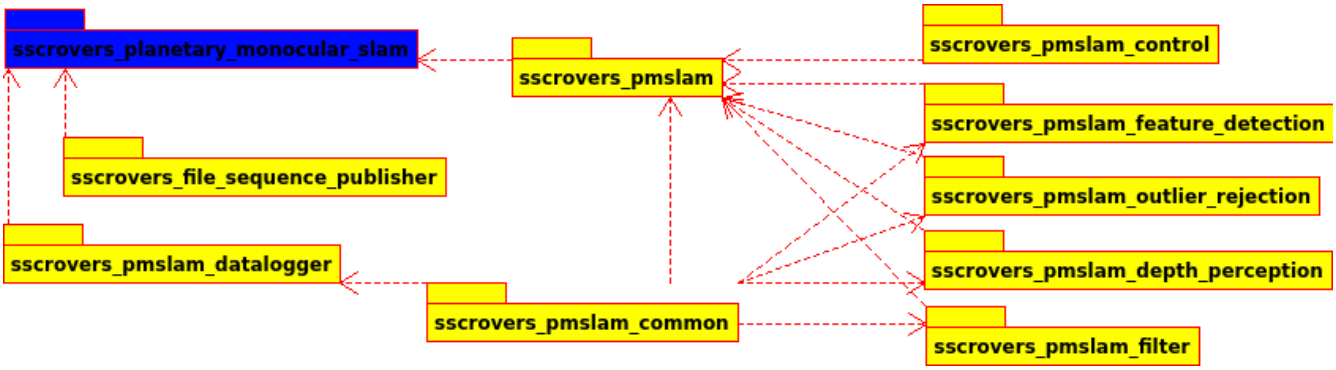
Units, state, position representations, coordinate system

4.2 Modules Description

MPMSlam project collected in the ROS stack (sscrovers_planetary_monocular_slam) contains set of modules. Each module is implemented as a different ROS package. The following diagram presents the modularized structure of the project:



Dependencies diagram and list of MPMSlam project packages are presented below:



- *sscrovers_file_sequence_publisher* is a helpful module independent of MPMSlam but used in the test and debug work. It allows to tunnelling the required input data for MPMSlam by

loading them from files. The node reads the sequence of images and position information and publish it as topics with image and odometry information. In that way this package simulates data form robot sensors.

- *sscrovers_pmslam_common* module contains common structures and methods for the all system. It contains descriptions of custom ROS messages, data types and global functions such as data display, conversion, storing to a file etc. Module code is placed inside sscrovers_pmslam_common namespace and built as a static library (pmslamcommon stored in file lib/libpmslamcommon.so). Other modules can link to this library by adding *target_link_libraries(<node_name> pmslamcommon)* command inside CMakeLists.txt file and include headers form <sscrovers_pmslam_common/*.h> directory for content which will be used in the source code. Namespace name and header files location was chosen that way to use the same naming convention for custom common code and auto-generated one.

Structures implemented in *sscrovers_pmslam_common* module:

- Afeature
- AfeatureSet
- RoverState
- SURFFeature
- SURFFeatureVector

sscrovers_pmslam package contains launch scripts (*.launch) for running MPMSlam as a whole project, and configuration files (*.yaml) collecting global (for whole project) as well as local (for concreate node) parameters to send it to the ROS Parameter Server before project will be launched. This package could be used for building all MPMSlam packages – all packages which are responsible for data processing (without sscrovers_pmslam_datalogger and sscrovers_pmslam_file_sequence_publisher which should be built independently).

sscrovers_pmslam_database (work in progress)

sscrovers_pmslam_control

sscrovers_pmslam_datalogger

sscrovers_pmslam_depth_perception

sscrovers_pmslam_feature_detection

sscrovers_pmslam_filter

sscrovers_pmslam_outlier_rejection

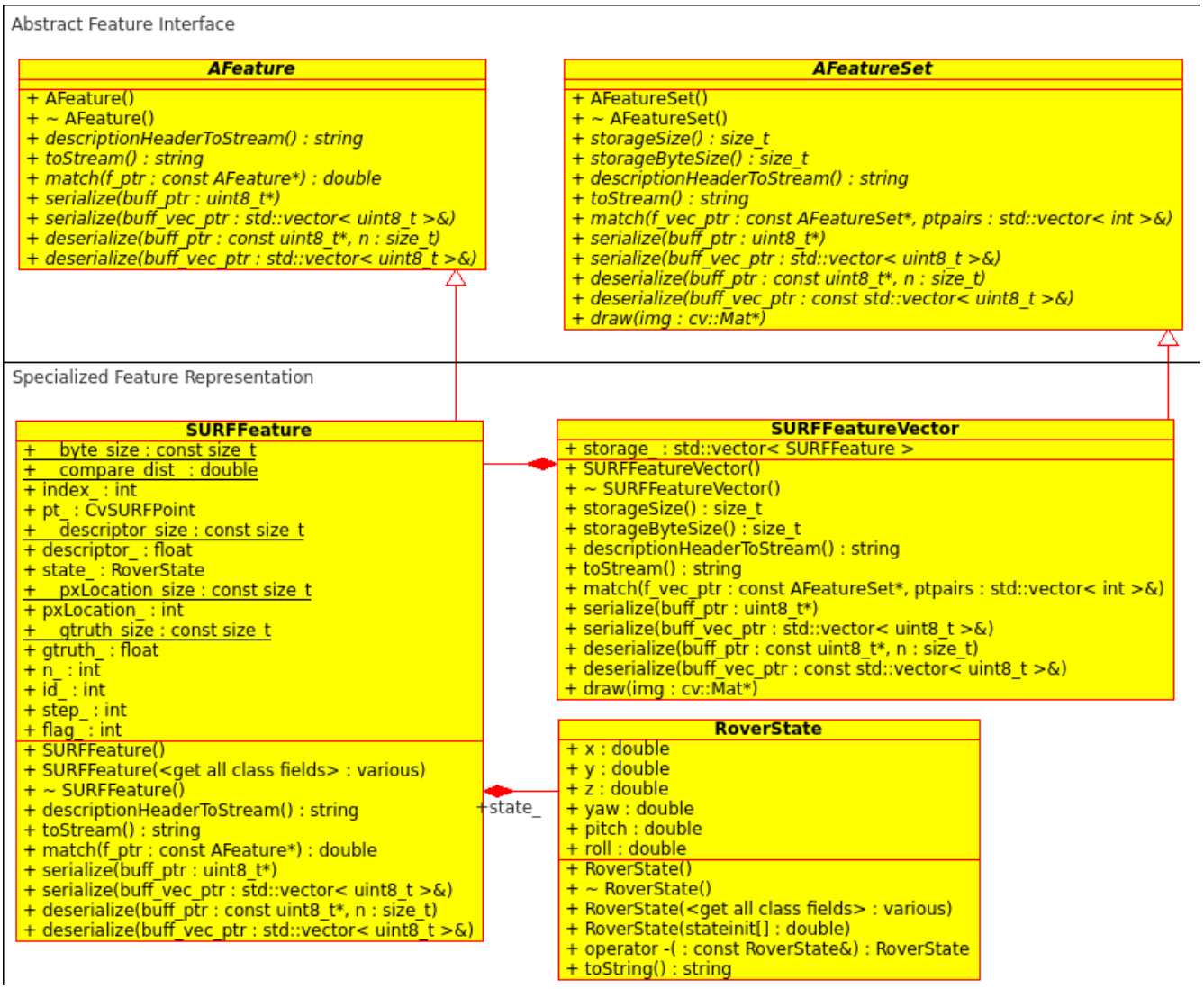
4.3 ROS messages.

All definitions of ROS messages, used in the project for exchanging data between ROS nodes, are collected in sscrovers_pmslam_common package. This approach allows the user to quickly find the specifications for each data structure and add new fields, which should be passed with standard ROS message (e.g. add step counter to image information, etc.). Below list presents a description of used messages (for more detailed information see file content and specific ROS message documentation):

- DynamicArray – message for passing multidimensional, serialized data as a linear byte buffer with additional description (dimensions of data, string description of serialized data type and header)
- ControlVector – control vector data which are represented by geometry_msgs/Point
- PtPairs – vector of geometry_msgs/Point's to corelate (by its id fields) currently extracted feature and feature database record
- IDPoint – representation of 3D geometry_msgs/Point with integer ID as a base type for vector in PairedPoints3D message
- PairedPoints3D – vector of IDPoints to exchanging 3D position extracted from and corelated (by id field) to feature in database
- PMSlamData – output data from PMSlam project. It contains 3D output map and current rover state
- Landmark – representation of the landmark for PMSlam output map
- Map3D – representation of PMSlam output map – optional representation with detailed information from EIF – not used.
- Points3D – deprecated (check it)
- Image (planned)
- Trajectory (planned)

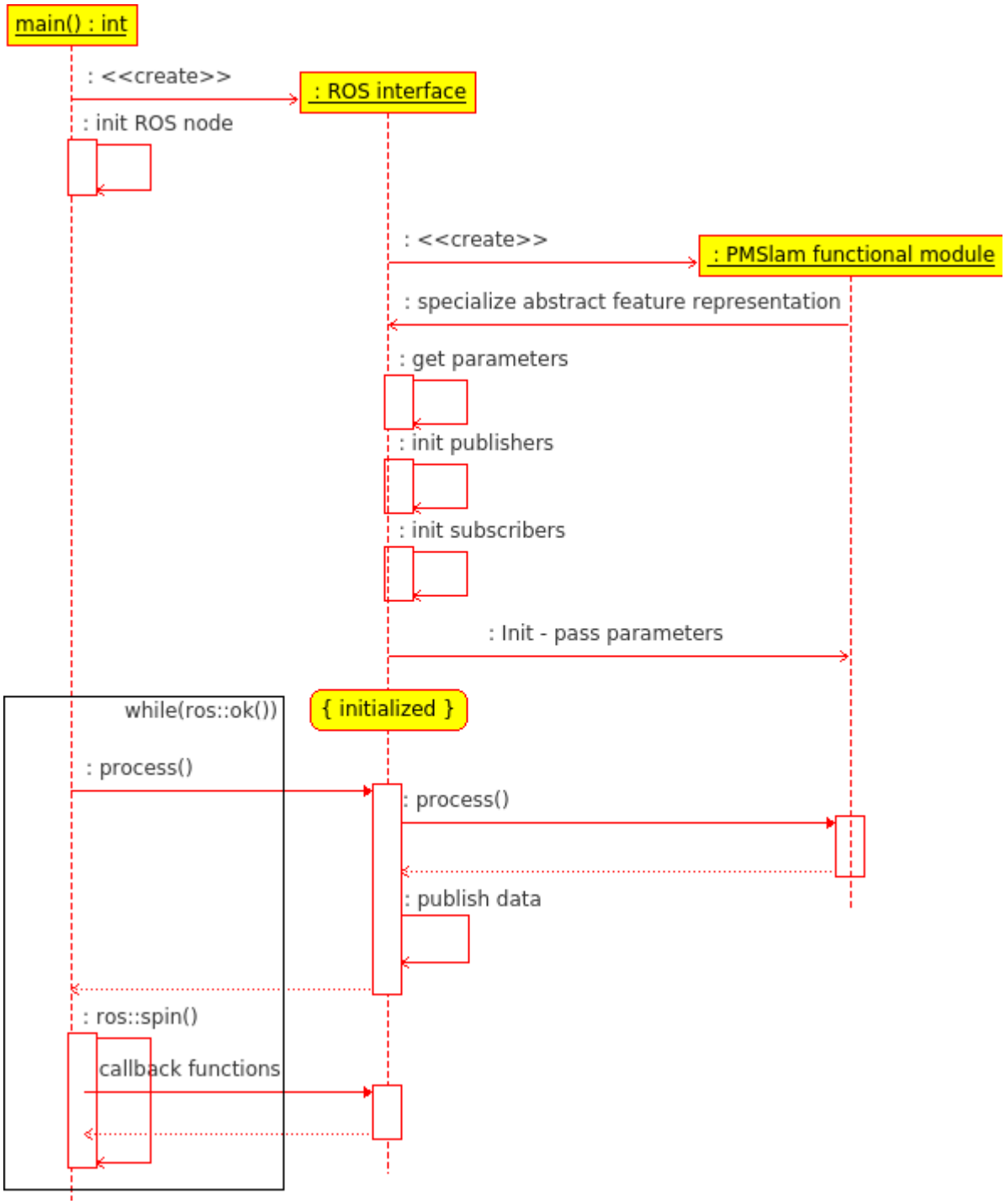
4.4 Features implementation

Generic Features Implementation was divided into two parts. Below diagram presents detailed structure of this representation. First part contains abstract representation of single Feature and Feature Set (abstract-interface classes AFeature and AFeatureSet). Second is a example specialization used in SURF features based system. Both abstract classes must be specialized by concreate implementation, and each of abstract class methods must be overwritten in final implementation. Below UML diagram presents dependencies between abstract and specific structures implementation for SURF features.



4.5 Interface separation

This assumption is realized by splitting the functionality of PMSlam processing algorithms and ros system interface to separated independent C++ objects. Each module contains separated definition and declaration of ros interface (in files <module_abbreviation>_core.h/cpp) and functional part of PMSlam (in files <function_name>_<module_abbreviation>.h/cpp) At the beginning of initialization (in constructor) of each module it is required to create object of PMSlam functional class. Diagram presented below shows the general way of division structure and realization of functionality inside single package.


main() : int
: <<create>>
: ROS interface
: init ROS node
: <<create>>
: PMSlam functional module
: specialize abstract feature representation
: get parameters
: init publishers
: init subscribers
: Init - pass parameters
while(ros::ok())
{ initialized }
: process()
: process()
: publish data
: ros::spin()
callback functions

4.6 Software Tools

5 Users Guide

It is very important part but was not planned in documentation template and was added recently. TODO ASAP.

6 Programmers Guide

It is a second very important part but was not planned in documentation template and was added recently.. TODO immediately after Chapter 5.

7 Testing

result comparison

8 Known Issues

no	Issue	Reason	Proposed solution	Priority
	impossible to measure time of message processing	step counter for every message is passing via header.stamp.nsec field	Create covered representation of all ROS messages in common and add to it step field	0

9 Future Work

10 References

1 Walter, M.R. and Eustice, R.M. and Leonard, J.J. *Exactly sparse extended information filters for feature-based SLAM* International Journal of Robotics Research vol.26 no.4 p.335-359 2007

2 Al-Milli, S. *Surrey Rover Autonomy Software & Hardware Test-bed (SMART)* Internal Report AI & Autonomy Group, Surrey Space Centre March 2011

3 Gao, Y. and Samperio, R. and Shala, K. and Cheng, Y. *Modular Design for Planetary Rover Autonomous Navigation Software using ROS* European Space Agency Acta Futura vol.5 no.1 p.9-16 2012

4 Beard, B. *Autonomous Systems Architecture Software Guide* Internal Report BAE Systems February 2011

5 Samperio, R. and Shala, K. and Pham, M.T. and Cheng, Y. and Gao, Y. *Pattern Design for robot environment modelling in planetary exploration* Internal Report of AI & Autonomy Group Surrey Space Centre, February 2011

6 *Robot Operating System Introduction* http://www.ros.org/wiki/ROS/Introduction September 2012

7 *The Player, Stage, Gazebo Project* http://playerstage.sourceforge.net September 2012

8 Gao Y. *Optical flow based techniques for ExoMars rover autonomous navigation*. European Space Agency Proceedings of iSAIRAS, LA, USA: Int. Symp. Artificial Intelligence, Robotics and Automation in Space (iSAIRAS) 2008

9 Shala K, Gao Y. *Review and analysis of localization and mapping techniques for planetary rovers* European Space Agency Proceedings of iSAIRAS, Japan: The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS) 2010