Prepared by
Sorin Adam
Approved by
Tom Simonsen

Document No.
FS 00 150 004 en
Document Date: 2017-05-09
Revision: E
Reference: -

SAFE: Message specification for obstacle detection

Contents

1 Introduction

This document aims to describe the communication interface between the main controller, e.g., Conpleks Robotech Robot software running on a Conpleks Robotech RT101 or RT501, and the obstacle detection component.

1.1 System Architecture

1.1.1 Hardware architecture

This document specifies the message format used over the Ethernet interface connecting the main controller with the obstacle detection system. Figure 1 presents the hardware overview of a robot using GNSS, IMU and vision for localisation. The Robot Controller provides the interfaces to the vehicle and the implement, the Vehicle Controller is in charge of localisation and mission execution, and the Vision helps both fine-tuning the localisation and providing obstacle detection information. The RTK GNSS module and IMU are connected via serial lines to the Vehicle Controller. The Robot Controller, Vision and Vehicle Controller are communicating using a Ethernet interface.

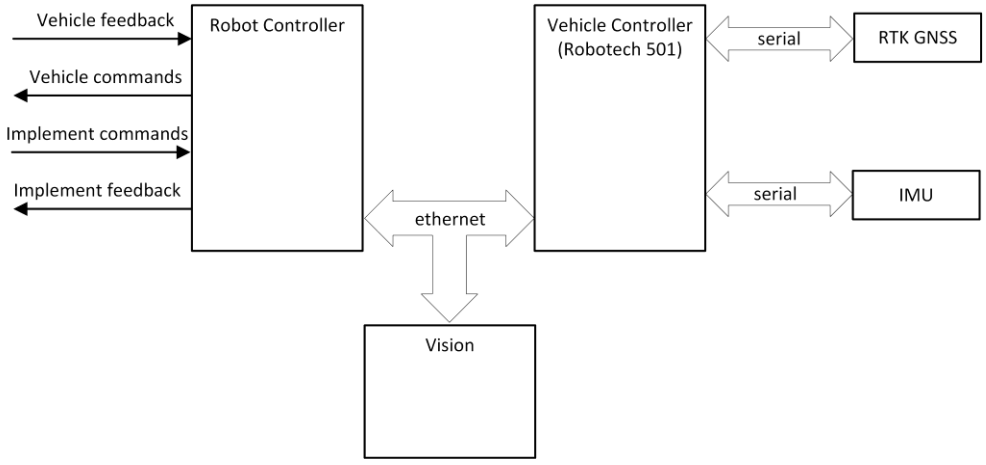

Figure 1: Hardware overview of a robot using GNSS, IMU and vision for localisation

1.1.2 Software architecture

Figure 2 presents a possible software architecture of a robot which uses GNSS, IMU and vision for localisation.

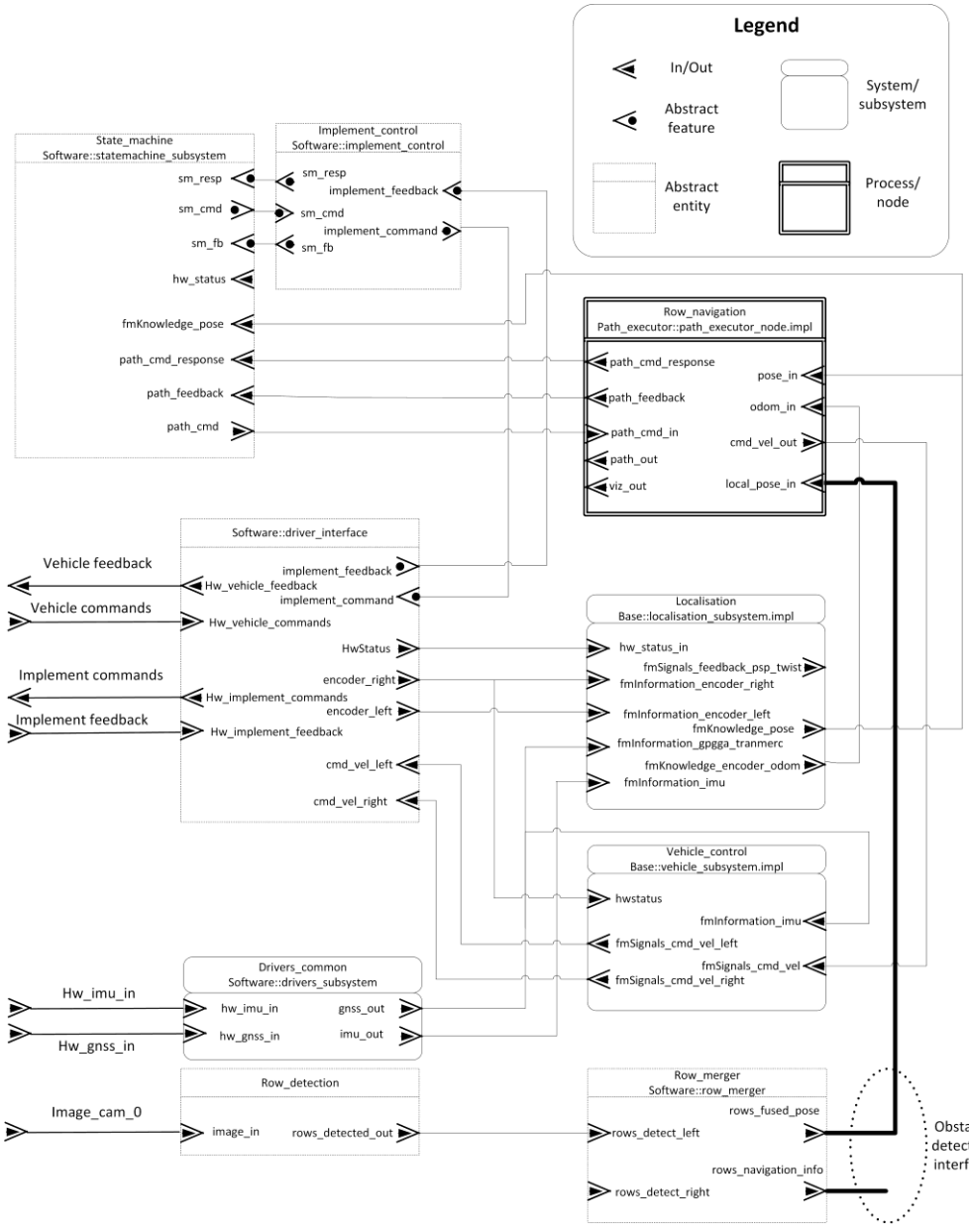

Figure 2: Software architecture overview of a robot which uses GNSS, IMU and vision for localisation

The architecture consists of several abstract entities (State_machine, Implement_control, driver_interface, row_detection and row_merger), subsystems or systems (localisation, vechicle_control, drivers_common) and a process/node (row_navigation). The different components are interconnected via physical input/output interfaces or abstract (logical) interfaces. This document specifies the obstacle detection interface (the thick lines in Figure 2).

2 Object detection elements

2.1 Obstacle virtual box

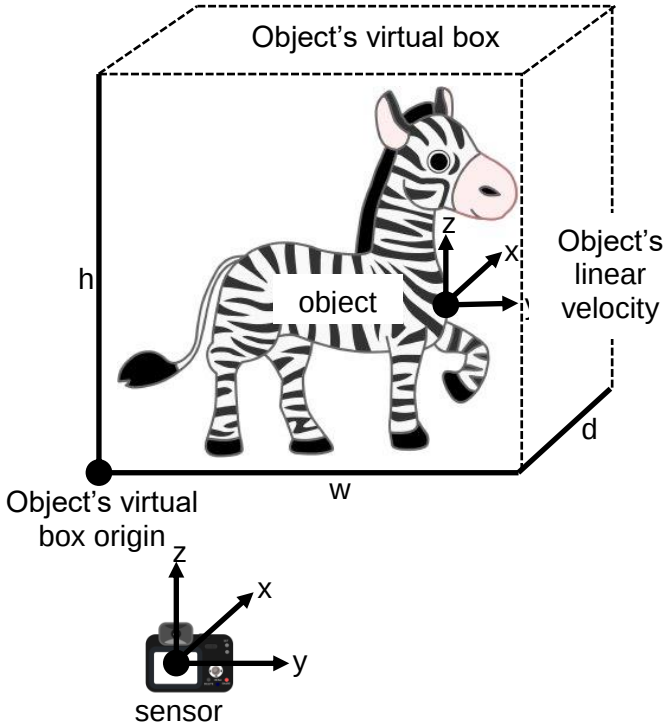

Figure 3 Obstacle virtual box, sensor coordinate system and obstacle linear velocity components

Figure 3 presents the main elements useful in performing obstacle detection. All measurements/estimations provided by the sensor are relative to the sensor, e.g., the distance x is the distance between the sensor and the object, and the x speed of the object is the relative linear speed on the x axis between the object and the sensor.

The object size is represented by the size of the virtual box enclosing the object (width, height, depth).

3 Message specification

3.1 Safe Object message

The sensor sends this message to inform about the objects it detects. The information for the detected objects is sent as arrays of data.

header	seq	1. Message sequence number
	stamp	2. Message timestamp
	frame_id	3. The ID of the frame
id		4. Object ID
type		5. Type of detected object
first_seen		6. Timestamp of first detection
det_confidence_level		7. Detection confidence level
position[]	x	8. Position of the object in Cartesian coordinate system (see Fig.3)
	y	
	z	
	confidence	9. Confidence of the position
rel_lin_vel[]	x	10. The linear component of the perceived relative object velocity; all fields must be 0 if not used
	y	
	z	
	quality	11. Quality of the speed estimation
size[]	w	12. Width of the virtual box surrounding
	h	13. Height of the virtual box surrounding
	d	14. Depth of the virtual box surrounding
	quality	15. Quality of the size estimation

3.1.1 Message fields description

1: message sequence number – incremented for every new message

2: message timestamp – the timestamp of the message in nanoseconds

3: frame ID – the frame ID of the sensor, used for frame transformations

4: object ID – the ID of the object; 0 if not used

5: object type – the type of object; 0 = not used, 1 = small object, 2 = large object

6: object first seen - the timestamp of the first detection of the message

7: detection confidence level – a float number between 0 and 1 (0 = lowest confidence, 1 = highest confidence); if the object detection confidence level is lower than a certain threshold, e.g. 0.6, means that the detection information is not reliable

8: object position in Cartesian coordinate system (relative to the sensor frame)

9: confidence of the position - a float number between 0 and 1 (0 = lowest confidence, 1 = highest confidence); if the object position confidence level is lower than a certain threshold, e.g. 0.6, means that the position detection information is not reliable

10: The linear components of the perceived object velocity (relative to the sensor frame); all fields containing a negative value (e.g.,-1) means that this information is not used (sensor not capable to provide this information)

11: Quality of the speed estimation; a negative value (e.g.,-1) indicates no estimation of the speed estimation is available

12-14: Object size – the dimensions of the virtual box enclosing the object (as shown in Fig. 3); a negative value (e.g., -1) means that the sensor is not able to provide info for that dimension

15: quality of the size - a float number between 0 and 1 (0 = lowest confidence, 1 = highest confidence); if the object size confidence level is lower than a certain threshold, e.g. 0.6, means that the size detection information is not reliable; a negative value (e.g., -1) means that this information is not available

3.1.2 Message format (ROS)

safe_sensor_msgs/SafeObject.msg

```
std_msgs/Header header
    uint32   seq
    time     stamp
    string   frame_id

int32 id    //the ID of the obstacle

int32 type // dynamic/static, positive/negative, human/animal, heat signature, etc.

float32 det_confidence_level   //confidence level of the detection

safe_sensor_msg/obj_position[] position //array of custom messages (object position)
   float64 x
   float64 x
   float64 z
   float32 confidence  //confidence level of the position

safe_sensor_msg/obj_lin_vel[] rel_lin_vel    //array of custom messages (velocity vector)
   float64 x    //x component of the speed
   float64 y    //y component of the speed
   float64 z    //z component of the speed
   float32 quality    //quality of the relative velocity estimation
```

```
safe_sensor_msg/obj_size[]   size //array of custom messages (object size)
   float64 w      //width
   float64 h      //height
   float64 d      //depth
   float32 quality    //quality of the size information
```

3.2 Safe Sensor Status message

header	seq	1. Message sequence number
	stamp	2. Message timestamp
	frame_id	3. The ID of the frame
sensor_status		4. Status

3.2.1 Message fields description

1: message sequence number – incremented for every new message

2: message timestamp – the timestamp of the message in nanoseconds

3: frame ID – the frame ID of the sensor, used for frame transformations

4: sensor status – the error state of the sensor (used for diagnosis); 0 = no error

3.2.2 Message format (ROS)

safe_sensor_msgs/SafeSensorStatus.msg

```
std_msgs/Header header
    uint32   seq
    time     stamp
    string   frame_id

int32 sensor_staus // Error state (related to sensor diagnostics)
```

3.3 Safe Safety Alert message

This message is used by the sensor to send safety alerts

header	seq	1. Message sequence number
	stamp	2. Message timestamp
	frame_id	3. The ID of the frame
zone_no		4. Zone number
confidence_level		5. Confidence level
alert_severity		6. Severity of the alert

3.3.1 Message fields description

1: message sequence number – incremented for every new message

2: message timestamp – the timestamp of the message in nanoseconds

3: frame ID – the frame ID of the sensor, used for frame transformations

4: zone number – the ID of the zone where a safety hazard has been detected by the sensor

5: detection confidence level – a float number between 0 and 1 (0 = lowest confidence, 1 = highest confidence); if the object detection confidence level is lower than a certain threshold, e.g. 0.6, means that the detection information is not reliable; a negative value (e.g., -1) means that this information is not available

6: severity of the alert – 0 = lowest severity

3.3.2 Message format (ROS)

safe_sensor_msgs/SafeSafetyAlert.msg

```
std_msgs/Header header
    uint32 seq
    time    stamp
    string  frame_id

int32 zone_no // Safety Critical Alert (based on configurable safety zones, system parameters)

float32 confidence_level   //confidence level of the alert

int32 alert_severity   //severity of the alert (e.g., warning, lift, stop, etc. )
```

3.4 Safe Sensor Management message

This message is used to configure the frequency the sensor sends update information for a specific object ID

header	seq	1. Message sequence number
	stamp	2. Message timestamp
	frame_id	3. The ID of the frame
object_id		4. Object ID
update_freq		5. Update frequency

3.4.1 Message fields description

1: message sequence number – incremented for every new message

2: message timestamp – the timestamp of the message in nano seconds

3: frame ID – the frame ID of the sensor, used for frame transformations

4: object ID – the ID of the object the message is applicable to; -1 means for all objects

5: update frequency in Hz – how often the information for that object is transmitted by the sensor

3.4.2 Message format (ROS)

safe_sensor_msgs/SafeSensorManagement.msg

```
std_msgs/Header header
    uint32 seq
    time     stamp
    string    frame_id

int32 object_id    //the ID of the object

int32 update_freq   //the update frequency of the message
```

4 Appendix – ROS message details

4.1 Message types and formats

The messages are presented several formats:

- Message overview where the fields of the message are described together with the overall format
- The roscpp message format
- The rospy message format

4.1.1 The roscpp message format

In general, a roscpp message format is:

```
package_name/msg
```

The `package_name` is the name of the package containing the message, and `msg` is the message.

For example, the `String` message from the `std_msgs` packages:

```
std_msgs/String
```

have to be included in the code in the following way:

```
#include <std_msgs/String.h>

...

std_msgs::String str;

...
```

The messages are composed of one or more field types:

```
fieldtype1 fieldname1

fieldtype2 fieldname2

fieldtype3 fieldname3
```

An example is:

```
int32 x

int32 y
```

For more details about the roscpp messages see http://wiki.ros.org/roscpp/Overview/Messages

4.1.2 The rospy message format

Similar with the roscpp case, a rospy message format is:

```
package_name/msg
```

The `package_name` is the name of the package containing the message, and `msg` is the message.

For example, the `String` message from the `std_msgs` packages:

```
std_msgs/String
```

have to be included in the code in the following way:

```
import std_msgs.msg

...

msg = std_msgs.msg.String()

...
```

Alternatively, it could be used as following:

```
from std_msgs.msg import String

...

msg = String()

...
```

The messages are composed of one or more field types:

```
fieldtype1 fieldname1

fieldtype2 fieldname2

fieldtype3 fieldname3
```

An example is:

```
int32 x

int32 y
```

For more details about the rospy messages see
http://wiki.ros.org/rospy/Overview/Messages

4.2 Field types

The following table presents the field types correspondence between roscpp and rospy:

Primitive Type	Description	C++	Python
bool	unsigned 8-bit int	`uint8_t`	`bool`
int8	signed 8-bit int	`int8_t`	`int`
uint8	unsigned 8-bit int	`uint8_t`	`int`
int16	signed 16-bit int	`int16_t`	`int`
uint16	unsigned 16-bit int	`uint16_t`	`int`
int32	signed 32-bit int	`int32_t`	`int`
uint32	unsigned 32-bit int	`uint32_t`	`int`
int64	signed 64-bit int	`int64_t`	`long`
uint64	unsigned 64-bit int	`uint64_t`	`long`
float32	32-bit IEEE float	`float`	`float`
float64	64-bit IEEE float	`double`	`float`
string	ascii string (4)	`std::string`	`string`
time	secs/nsecs unsigned 32-bit ints	`ros::Time`	`rospy.Time`
duration	secs/nsecs signed 32-bit ints	`ros::Duration`	`rospy.Duration`

For more details see http://wiki.ros.org/msg.