Software Requirements Specification

Firefighter Indoor Navigation

Client: Mostafa Daneshgar Rahbar

GTA: Samira Taghavi

Team Members: Thomas Anter
Jordan Shimel
Nawar Mikha

Revision History

Date	Version	Authors	Comments
9/22/19	Version 1.0	Thomas Anter Nawar Mikha Jordan Shimel	First Draft
12/5/19	Version 2.0	Thomas Anter Nawar Mikha Jordan Shimel	Added software and hardware interface diagrams, changed some requirements, formating, added to user characteristics, and more

Document Approval

Signature	Printed Name	Title	Date
	Samira Taghavi	Graduate Teaching Assistant	12/5/19
	Mostafa Daneshgar Rahbar	Client	12/5/19
	Jordan Shimel	Testing Lead / Integration Lead	12/5/19
	Nawar Mikha	Team Lead / Presentation Lead /UI Lead	12/5/19
	Thomas Anter	Documentation Lead / Back-End Lead / Front-End Lead	12/5/19

Table of Contents

List of Figures

1 Introduction

This document serves several purposes; it is meant to: outline the design of the product, act as a liquid contract between the client and the production team, and as a reference for future software maintainers. The SRS is viewed as a living document, new versions with some amendments will likely be produced throughout the products production. The primary information in this document includes: a general system overview, process behind the requirements development, the requirements themselves, and an overview of how the system functions.

Having a well produced SRS can help avoid errors during the build phase; build complications being more costly the later in the project that they are discovered. The design team will be able to begin their work immediately upon delivery of this document. It's also a good reference for the entirety of the product's lifecycle, especially when implementing new functionality.

The other large benefit of this document is that it formally outlines the expectations of what the product is for all parties involved. Any possible disputes between the client and the production team can be resolved by referencing the various requirements sections. The rest of this portion of the document includes general background information on the system and the document itself.

1.1 Purpose

The purpose of this document is to describe the functionalities of the Firefighter Indoor Navigation system, and how it fulfills the needs of the stakeholders, Mostafa Daneshgar Rahbar, and Samira Taghva. The following sections of this document will serve as a contract of agreement for our client to assure the requirements are understood and carried out. It will also go into detail to describe the constraints, dependencies, analysis, requirements, and functionalities, both functional and nonfunctional, associated with the project.

1.2 Scope

The goal and scope of the Firefighter Indoor Navigation system is to build a working distributed system to create a real time digital mapping of the route taken by a firefighter and display it in a GUI within an application window. The mapping will be done by creating a point cloud that will display the environment, the current location of the firefighter in that environment, the path they took to get to that location, and the orientation of the camera along that path. The system will support up to two cameras at the same time. There will be two computing devices in the system. One unit will be a depth-finding camera connected to an up board. This will be referred to as the remote unit. There will also be a main console, referred to as the base unit.

The remote unit consists of the depth finding camera and the UP board, which will be connected via USB. The camera requires a direct connection to transmit the data being received. The UP board has no built in Wi-Fi capabilities, so there will be a Wi-Fi module connected as well. The objective of the remote unit is to collect the depth information where the camera is pointing, along with a record of the camera's location and orientation, to convert that live data stream to a series of ROS messages, and to publish those ROS messages via point-to-point Wi-Fi to the base unit. The remote unit is intended to be mounted to the gear of a firefighter and is needed to track the location of that firefighter.

The base unit will be a laptop. The objective of this unit is to receive a live stream of ROS messages via point-to-point Wi-Fi from the remote unit and perform SLAM on that data. The SLAM package's API's will create a point cloud map from that data, and output both that point clouds, and a live video feed from that camera, on an active GUI on the laptop. The GUI will display one map at a time, but there will be tab options to switch between maps. The point cloud visualization will support panning and scanning operations in the 3D maps. The location of the cameras the path taken to get to that location, and the orientation of the camera on that path, will all be displayed in the 3D sparse map.

The goal of this software is to provide fire departments with a better alternative to GPS as it pertains to tracking workers in an unfamiliar

building. This will allow firefighters to refer to a digital map created in real time as they explore a building. The map will be created with the remote unit, the base unit operator outside the building, will be able to see the location of a firefighter with the remote unit in the building and be able to navigate that individual more efficiently.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
API	Application program interface
Bag	A file format for storing ROS messages
Base Unit	Laptop with control software. “base station”, “main unit”, and “main station” will be used synonymously
FIN	Firefighter indoor navigation
FFMPEG	Video capture and encoding package
GUI	Graphical user interface
IMU	Inertial Measurement Unit
Messages	Structured data file
Nodes	Executable that can communicate with other ROS nodes
OpenCV	Open computer vision
OpenGL	Open graphics library
ORB-SLAM2	Real-time SLAM library for Monocular, Stereo and RGB-D cameras
Point Cloud	Collection of data points defined by a given coordinate system
Priority	Scale: (1 - 3) (1 = low, 2 = mid, 3 = high)
Remote Unit	UP board connected to a camera
ROS	Robotics operating system

Roscore	Or ROS Master, is a collection of nodes that are a prerequisite for ROS based systems.
QT	GUI development tool
SLAM	Simultaneous localization and mapping
Stereo Camera	Camera with two lenses that uses binocular vision to measure points of depth
UP Board	Small computer
Ubuntu	Linux based operating system

Figure 1

1.4 Overview

This document is intended to outline the entire system and its features. It begins by providing some introductory information about the document itself. Following that, is a section that illustrates a broad overview of the system. Then in section 3, the detailed requirements for the system are defined. In the final section, there is a Data flow diagram that represents how data is flowing through the system.

The third section is where the bulk of the content of this document is located. All major functional, non-functional and external interface requirements are located in the specific requirements portion of the document. These lists of requirements were developed by the team members and the client in collaboration.

2 General Description

This section is meant to provide insights into why the system is being constructed in its chosen manner. First it explains where this product fits into the marketplace and how it compares with similar projects. Then the general functions of the product are described. Following that, it outlines

the basic attributes of the end users and how those factors were taken into consideration for building the requirements list. Overall constraints are discussed in the next section. In the final portion, assumptions and dependencies are explored.

2.1 Product Perspective

Many similar projects have already been produced; many of them vary somewhat, in that they use different and often times more powerful hardware, as well as having a different set of requirements. When evaluating the system's performance, the recorded data in the reports of similar projects, that use the same hardware as this system, can be used for comparison. Correctly mapping the data points seems to be one of the more involved tasks that will require an efficient algorithm to enable the system to function in real-time, here is where examining other systems may prove to be the most useful.

The Wi-Fi adapter for the up board is listed as having a maximum bandwidth of 150 megabits-per-second. At 24 bits per pixel (for red, blue, green, pixel data), setting the camera to 20 frames-per-second and a resolution of 640x480, will produce around 147.5 Megabits-per-second of data. Data transfer over the network shouldn't be a problem, but we'll have to observe in practice how much bandwidth the system will actually have and if it can carry the load of the camera data produced in real-time. If there are issues in this regard, the frame rate and resolution will have to be adjusted and tested to ensure that the system is able to functionally operate in real-time.

2.2 Product Functions

The main functions that the software will perform are as follows:

- Collect data from a remote unit equipped with a stereoscopic depth camera and inertial monitoring unit

- Transmit image depth data and motion data to a base station for processing and analysis
- Allow the display of both the raw image data and point cloud maps created from the data on the base station
- Integrate multiple map viewers to assist with the overall navigation task

2.3 User Characteristics

Although firefighters will be the ones equipped with the cameras and up boards, the actual software interface will be used by a team coordinator, either in the fire truck or at the fire station. Firefighter team coordinators are whom the system is designed to work for; it is assumed that no one else will be using this product. Given that only this group of users will actually interact with the system, there doesn't seem to be a need to build distinct roles, such as an administrator. The only other role to consider, which is trivial in scope, is that of the person starting the remote camera units (the actual firefighters), but this user should only have to start the device once the required operating system and software packages are installed, and the settings are configured.

To build a product that will be as useful as possible, what information and functionality a team coordinator will be concerned with, needs to be considered. This individual will need to be able to get real-time data from the software so that it can be relayed to the firefighters immediately through their standard radio system, as their job is time critical. This is the main user the team has considered in its use cases for developing the requirements of the system. The coordinator will be assisted with his/her role as navigator with a few different maps in the base unit application. This individual will have a dense map that shows a full detailed render of the environment of the firefighter, a sparse map for the position of the firefighter, and an overhead view of the sparse map to create a sort of grid of the building.

2.4 General Constraints

Possible constraints to this system include:

- Limitations on the available bandwidth and reliability of a Wi-Fi connection, possibly necessitating the imposition of limits on the fidelity of data collected from the camera. If the signal is too weak a wire connect will be used for proof of concept.
- Processing limitations on the base station, which may require a reduction in the resolution of the point cloud, in order to ensure real time operation.
- Limitation on the number of cameras that can be connected to the base unit at once without slowing down the base unit's real-time data processing.

2.5 Assumptions and Dependencies

Remote unit assumptions:

- The remote unit hardware (up board) is capable of functioning with the required hardware and software to successfully capture the required depth and motion data
- The transmission hardware is capable of providing sufficient bandwidth and latency to enable transmission of the data from the remote unit to the base unit.

Base unit assumptions:

- The transmission hardware is capable of providing sufficient bandwidth and latency to enable receipt of data from at least two remote unit instances.
- The base station hardware is capable of real time processing of the depth and motion data into a 3D point cloud.

Software dependencies:

- DBoW2
  - Library for transforming images into bag-of-words (Requirement of ORB-SLAM2)
- Eigen3
  - Linear algebra library (Requirement of g2o)
- FFMPEG
  - Video capture and encoding (Requirement of OpenCV)
- G2o
  - Graph optimization library (Requirement of ORB-SLAM2)
- Glew
  - Extension for OpenGL (Requirement of Pangolin)
- GLFW-
  - Extension for OpenGL
- GTK+
  - Toolkit for creating GUI (Requirement of OpenCV)
- Intel RealSense SDK 2.0
  - API for interfacing with D435 camera
- OpenCV 4.1.1
  - Computer vision library for image processing
- OpenGL
  - Graphics API (Requirement of Pangolin)
- ORB-SLAM2
  - Real-Time SLAM for camera data
- Pangolin
  - Visualization and UI (Requirement of ORB-SLAM2)
- PCL-
  - Point cloud processing library
- Qt5 -
  - GUI development tools
- XDo -
  - C++ library for window manipulation

3 Specific Requirements

Through rigorous discourse between the client and the team members, a detailed list of system requirements was developed. To ensure that these

requirements meet professional standards, each has been checked to be: correct, traceable, unambiguous, verifiable, prioritized, complete, consistent, understandable, non-prescriptive, and concise.

3.1 External Interface Requirements

In this section, the various system interface requirements are outlined. A simple design that is easily accessible was the driving idea behind the GUI. Due to the time critical nature of firefighting, users of the product will need to be able to access various features quickly without a complex toolbar or drop down menus that could slow down the user experience by even a fraction of a second. In addition, a complex system such as this, also has many software and hardware interface requirements. If one library is one version too old or new it could cause a failure of the entire system. Software and hardware requirements were thoroughly researched in order to avoid catastrophic compatibility errors in the production phase.

3.1.1 User Interfaces

Once the data is sent via the point-to-point Wi-Fi, the base station will perform SLAM on that live data feed. QT will be used to create a GUI and data from the results of SLAM will be visualized in that GUI. The user will be able to set the configuration, test the camera, and send its data on the remote unit. The user will have a video stream in the left half of the application window, and 4 tabs on the right within the base unit. The tabs will be for the sparse point cloud map, the overhead view of the sparse map, the dense point cloud map, and a setting page. A final GUI would look similar to the following layouts for the base and remote units respectively:

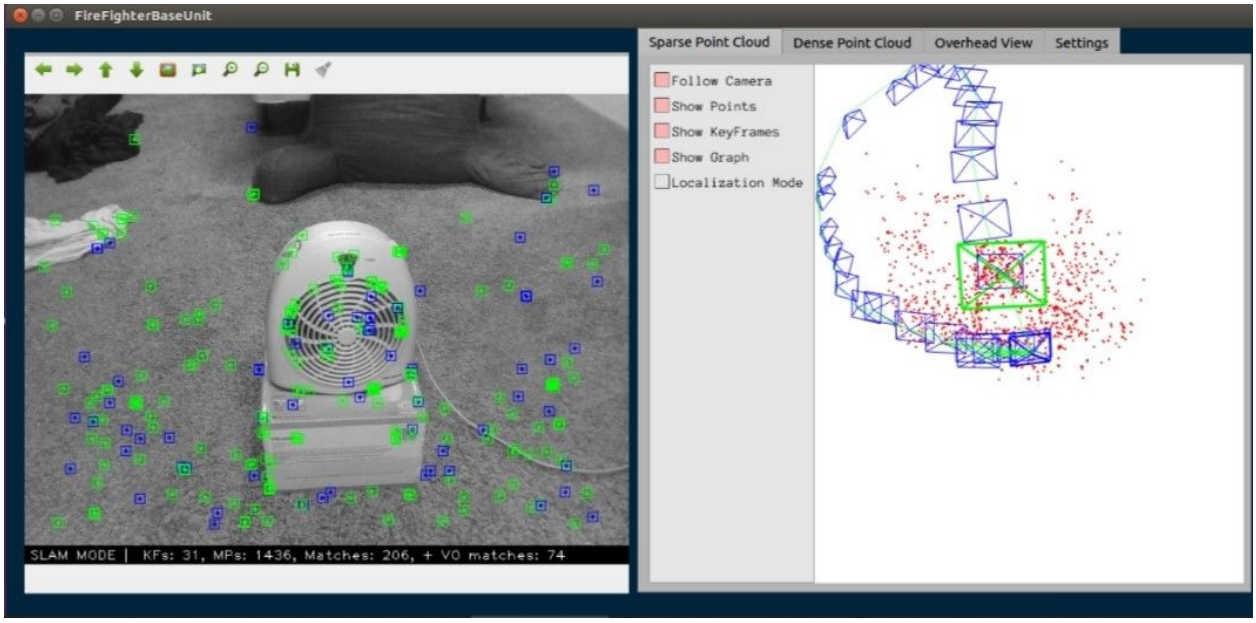

Figure 2

Figure 3

3.1.2 Hardware Interfaces

As a distributed system, there are multiple levels of hardware involved in this project. The five most vital components are the base station laptop, the up board, the stereo camera, screen display, and Wi-Fi adapter for the up board.

The base station will be a portable PC running the same version of Linux as the remote unit, Ubuntu 16.04. The base station is responsible for receiving video, depth and IMU data from the remote unit, performing SLAM on that data to generate a point cloud, and displaying that point cloud and video stream in a GUI. The base station will receive this data via a point-to-point Wi-Fi connection with the remote unit. The final base station to be used in deployment can be any machine that will run Ubuntu 16.04, and that is powerful enough to perform the SLAM operations on the camera data in real-time. For the purposes of testing, and practical demonstration, the base station will be a student laptop. The entire system's components and how they are connected are shown in the following graphic.

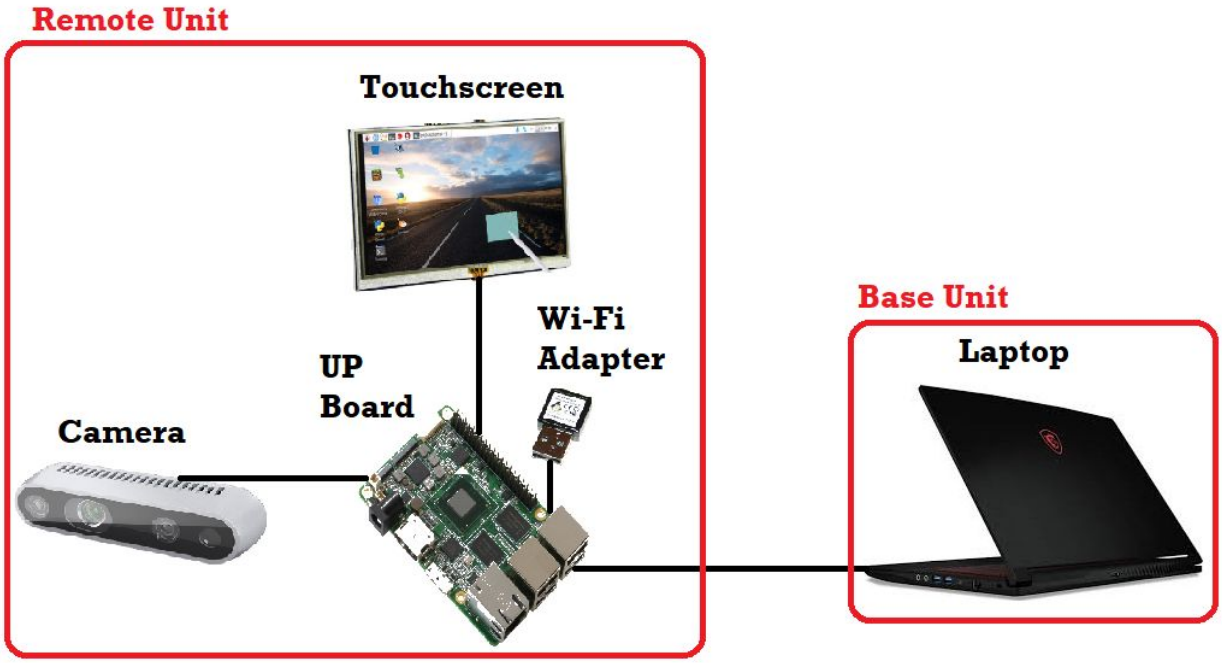

Figure 4

The laptop being used as the base station for development purposes has the following specifications:

- Intel® Core™ i5-8300H Processor
    - Quad core
    - 64-bit architecture
    - 8MB cache
    - Up to 4.00Ghz operating frequency
- 8GB DDR4
    - 2400Mhz operating frequency
- NVidia GTX 1050Ti Max-Q graphics processor

The remote unit and the camera work in tandem and are connected directly via USB 3.0. The remote unit itself is a single board computer called an up board. The specifications of the up board are as follows:

- Intel® Atom™ x5-Z8350 Processor
    - Quad core
    - 64-bit architecture
    - 2MB cache
    - Up to 1.92Ghz operating frequency
- 2GB DDR3L memory
    - 1600Mhz operating frequency
- 32GB eMMC flash memory
- HDMI 1.4a
- 1x Gigabit ethernet RJ-45
- 4x USB 2.0, 1x USB 3.0 OTG
- 5V DC-in
- Measures 85.6 mm x 56.5 mm

Connected to the up board is the Intel® RealSense™ Depth Camera D435i. The camera is able to capture RGB video, depth video and IMU data. The specifications of the RealSense D435i are as follows:

- Depth
    - Active IR stereo
    - Output resolution and frame rate
        - Up to 1280 x 720, at up to 90 fps

- RGB
    - Sensor resolution and frame rate
        - Up to 1920 x 1080, at up to 30 fps
- Major components
    - Camera module
        - Intel RealSense Module D430 + RGB Camera
    - Vision Processor Board
        - Intel RealSense Vision Processor D4
- Connection
    - USB-C 3.1 Gen 1
- Dimensions
    - 90 mm x 25 mm x 25 mm

A 5 inch Elecrow LCD screen will be used to interface with the up board. The screen has the following specifications:

- 5.0" TFT LCD Module
- Resolution:800x400
- Touch Screen Type: Resistive LCD driver IC: ILI9486L
- Refresh rate: 60Hz
- LCD Size: 121.11mm*77.93mm
- Micro USB: get 5V power (not needed when connected by GPIO)
- GPIO (13*2): get 5V power & touch function
- HDMI interface: for HDMI transmission

The up board will connect to the network setup in the base unit through a TPE-N150USB Wi-Fi adapter. The adapter's specifications are as follows:

- Chipset
    - AR9271
- Data Rate
    - 150Mbps @ 400GI Max
- Dimensions
    - 30(L)x 17(W) x 8.5(H) mm
- Compatible With
    - MS Windows XP/Vista/7and GNU/Linux/OpenBSD 6.3+
- Security
    - 64/128/152-bit WEP encryption

    - 802.1x authentication
    - AES-CCM & TKIP encryption
    - WPA/WPA2
- Operation Distance
    - Outdoor: with 802.11n can get about 250m (about .15 miles) in good conditions

3.1.3 Software Interfaces

Qt, a graphics framework for the c++ language will be used to create the software interface on both the remote and base units. ORB-SLAM2, one of the main libraries that will be used in the development of these applications has many functions that will be implemented into the base unit by default, unless otherwise specified. On the remote unit, the user will be able to set the configuration, test the feed, and send the data. On the base unit the user will have a video stream in the left half of the application window, and 4 tabs on the right. The tabs will be for the sparse point cloud map, the overhead view of the sparse map, the dense point cloud map, and a setting page. The following diagram shows the planned software interface for the system to be able to execute these planned functionalities:

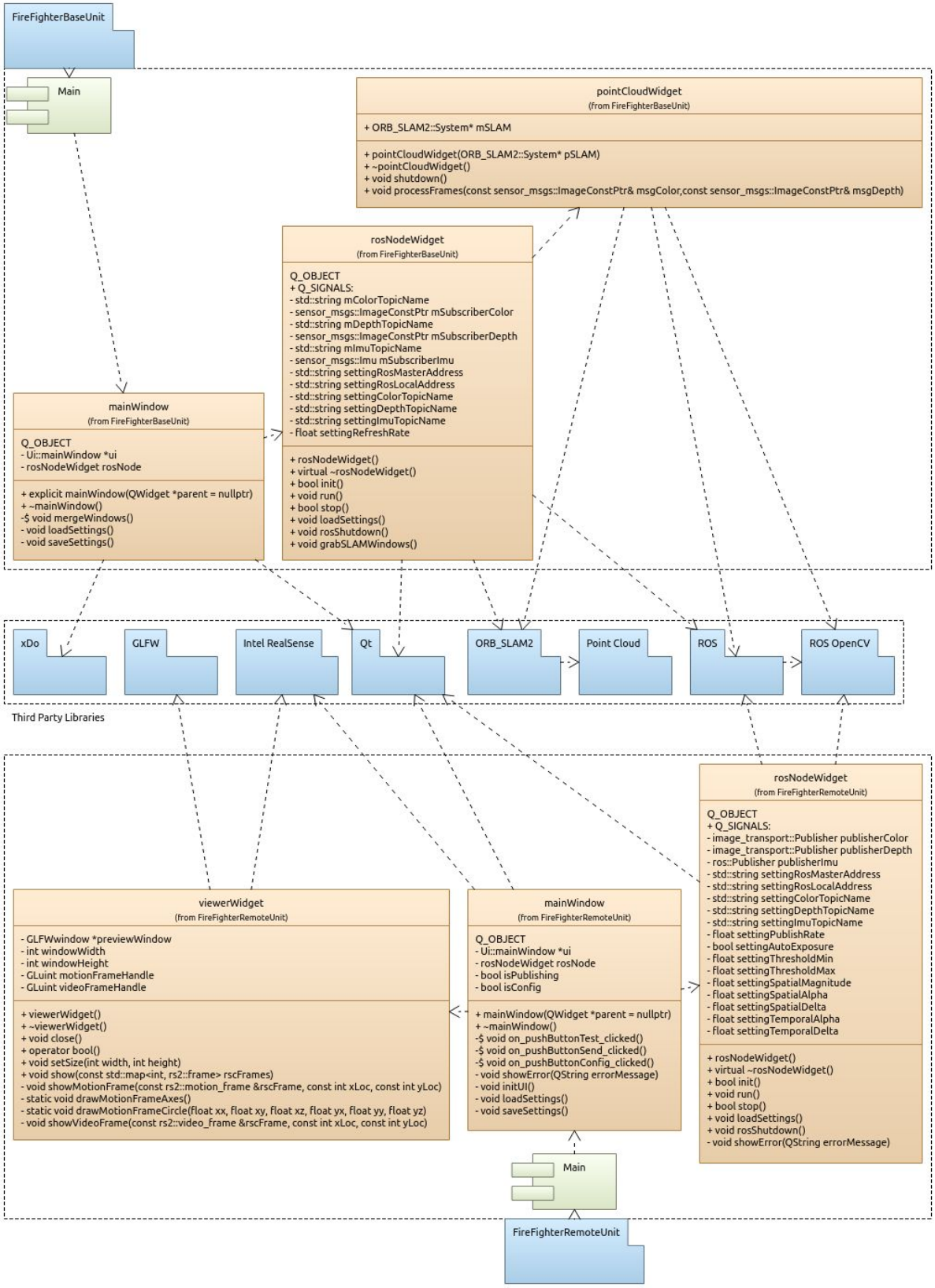
FireFighterBaseUnit
Main
pointCloudWidget
(from FireFighterBaseUnit)
+ ORB_SLAM2::System* mSLAM
+ pointCloudWidget(ORB_SLAM2::System* pSLAM)
+ ~pointCloudWidget()
+ void shutdown()
+ void processFrames(const sensor_msgs::ImageConstPtr& msgColor,const sensor_msgs::ImageConstPtr& msgDepth)
rosNodeWidget
(from FireFighterBaseUnit)
Q_OBJECT
+ Q_SIGNALS:
- std::string mColorTopicName
- sensor_msgs::ImageConstPtr mSubscriberColor
- std::string mDepthTopicName
- sensor_msgs::ImageConstPtr mSubscriberDepth
- std::string mImuTopicName
- sensor_msgs::Imu mSubscriberImu
- std::string settingRosMasterAddress
- std::string settingRosLocalAddress
- std::string settingColorTopicName
- std::string settingDepthTopicName
- std::string settingImuTopicName
- float settingRefreshRate
+ rosNodeWidget()
+ virtual ~rosNodeWidget()
+ bool init()
+ void run()
+ bool stop()
+ void loadSettings()
+ void rosShutdown()
+ void grabSLAMWindows()
mainWindow
(from FireFighterBaseUnit)
Q_OBJECT
- Ui::mainWindow *ui
- rosNodeWidget rosNode
+ explicit mainWindow(QWidget *parent = nullptr)
+ ~mainWindow()
-$ void mergeWindows()
- void loadSettings()
- void saveSettings()
xDo
GLFW
Intel RealSense
Qt
ORB_SLAM2
Point Cloud
ROS
ROS OpenCV
Third Party Libraries
rosNodeWidget
(from FireFighterRemoteUnit)
Q_OBJECT
+ Q_SIGNALS:
- image_transport::Publisher publisherColor
- image_transport::Publisher publisherDepth
- ros::Publisher publisherImu
- std::string settingRosMasterAddress
- std::string settingRosLocalAddress
- std::string settingColorTopicName
- std::string settingDepthTopicName
- std::string settingImuTopicName
- float settingPublishRate
- bool settingAutoExposure
- float settingThresholdMin
- float settingThresholdMax
- float settingSpatialMagnitude
- float settingSpatialAlpha
- float settingSpatialDelta
- float settingTemporalAlpha
- float settingTemporalDelta
+ rosNodeWidget()
+ virtual ~rosNodeWidget()
+ bool init()
+ void run()
+ bool stop()
+ void loadSettings()
+ void rosShutdown()
- void showError(QString errorMessage)
viewerWidget
(from FireFighterRemoteUnit)
- GLFWwindow *previewWindow
- int windowWidth
- int windowHeight
- GLuint motionFrameHandle
- GLuint videoFrameHandle
+ viewerWidget()
+ ~viewerWidget()
+ void close()
+ operator bool()
+ void setSize(int width, int height)
+ void show(const std::map<int, rs2::frame> rscFrames)
- void showMotionFrame(const rs2::motion_frame &rscFrame, const int xLoc, const int yLoc)
- static void drawMotionFrameAxes()
- static void drawMotionFrameCircle(float xx, float xy, float xz, float yx, float yy, float yz)
- void showVideoFrame(const rs2::video_frame &rscFrame, const int xLoc, const int yLoc)
mainWindow
(from FireFighterRemoteUnit)
Q_OBJECT
- Ui::mainWindow *ui
- rosNodeWidget rosNode
- bool isPublishing
- bool isConfig
+ mainWindow(QWidget *parent = nullptr)
+ ~mainWindow()
-$ void on_pushButtonTest_clicked()
-$ void on_pushButtonSend_clicked()
-$ void on_pushButtonConfig_clicked()
- void showError(QString errorMessage)
- void initUI()
- void loadSettings()
- void saveSettings()
Main
FireFighterRemoteUnit

Figure 5

3.2 Functional Requirements

This section outlines the specific functional requirements of the system. The requirements are backwards dependent, but not necessarily on all of the previous requirements The following table is provided as a quick reference to the requirements that are expanded upon in greater detail afterwards.

Numerical Name	Descriptive Name
FR-3.2.1	Video Feed Test on Remote Unit
FR-3.2.2	Publishing Messages from Video to Topic
FR-3.2.3	Subscribe to Messages from the Video Topic
FR-3.2.4	Base Unit Video Stream
FR-3.2.5	Base Unit Top Down 2D Sparse Point Cloud
FR-3.2.6	Base Unit 3D Sparse Point Cloud
FR-3.2.7	Base Unit 3D Dense Point Cloud
FR-3.2.8	Remote Unit Settings
FR-3.2.9	Base Unit Settings

Figure 6

3.2.1 Video Feed Test on Remote Unit

FR-3.2.1	Video Feed Test on Remote Unit	
Purpose: To check if the firefighters camera is streaming data.		
Desc: When the "Test" button is clicked in the main menu GUI on the remote unit, if the camera is receiving data, the live video feed from the camera will be displayed on the remote unit.		
Input: Firefighter selects test button on remote unit GUI.		
Output: Video feed from that camera is displayed on the remote unit GUI.		
Error Handling: When the test button is selected, if the camera isn't connected, a message will be sent to the user indicating such.		
Version: 1.0	**Date:** 10/2/2019	**Priority:** 2

3.2.2 Publishing Messages from Video to Topic

FR-3.2.2	Publishing Messages from Video to Topic
Purpose: Send camera data from the remote unit to the base unit for processing.	
Desc: When the "Send" button in the main menu of the remote unit is pressed, the data being received from the camera will be converted to	

ROS messages and published to a ROS topic using a ROS publisher node on the remote unit.		
Input: Firefighter selecting the send button.		
Output: Notification to the user that the remote unit is now publishing to a topic, send button turns green, other two buttons become hidden.		
Error Handling: If the messages aren't being sent a test will catch the error and output an alert.		
Version: 1.0	**Date:** 10/2/2019	**Priority:** 2

3.2.3 Subscribe to Messages from the Video Topic

FR-3.2.3	Subscribe to Messages from the Video Topic
Purpose: Receive camera data from the remote unit on the base unit for processing.	
Desc: If the remote unit is sending data and both units have been configured correctly, then after the base unit application has been executed it will subscribe to the topic where the camera data is being published as messages enabling it to process this data for the views in the base unit GUI.	
Input: Launching base unit	
Output: GUI displaying incoming data.	

Error Handling: If the messages aren't being received, the application will still launch, but will say that no key frames are being received in the video stream.		
Version: 1.0	**Date:** 10/2/2019	**Priority:** 1

3.2.4 Base Unit Video Stream

FR-3.2.4	Base Unit Video Stream
Purpose: Interface for the Team coordinator to see the live video feed.	
Desc: Incoming messages are processed and displayed as a video stream. This will allow the operator to view the live camera feed generated by the remote unit's data.	
Input: Launching base unit, remote unit messages being subscribed to.	
Output: GUI displays live camera view in the left hand side of the base unit.	
Error Handling: If the messages aren't being received, the application will still launch, but will say that no key frames are being received in the video stream. If the camera is moved to quickly, it will lose track of its frame of reference and output that its off track and trying to reinitialize.	

Version: 1.0	**Date:** 10/2/2019	**Priority:** 3

3.2.5 Base Unit Top Down 2D Navigation Point Cloud

FR-3.2.5	Base Unit Top Down 2D Navigation Point Cloud
Purpose: Interface for the Team coordinator to to access the firefighter's location and surroundings.	
Desc: This will allow the operator to view the physical surroundings of the firefighter in a 2D grid view of the point cloud map. This will also show the firefighter's position, orientation, and path.	
Input: Launching base unit, remote unit messages being subscribed to.	
Output: GUI displays a 2D top down sparse point cloud with camera position.	
Error Handling: If the messages aren't being received the application will still launch, but will say that no key frames are being received in the video stream. If the camera is moved to quickly, it will lose track of its frame of reference and output that its off track and trying to reinitialize.	
Version: 1.0	**Date:** 10/2/2019	**Priority:** 3

3.2.6 Base Unit 3D Sparse Point Cloud

FR-3.2.6	Base Unit 3D Sparse Point Cloud
Purpose: Interface for the Team coordinator to access the firefighter's location and surroundings. Includes panning, scanning, and zooming	

functions to allow the operator to manipulate the view of the sparse point cloud map.
Desc: This will allow the operator to view the physical surroundings of the firefighter in a sparse 3D view of the point cloud map. This will also show the firefighters position, orientation, and path.
Input: Dragging the screen from various directions with the right and left clicks on the mouse and moving the scroll button on the mouse.
Output: A sparse 3D map of the surrounding area with the capability to display changes with the mouse.
Error Handling: If the messages aren't being received' the application will still launch, but will say that no key frames are being received in the video stream. If the camera is moved to quickly, it will lose track of its frame of reference and output that its off track and trying to reinitialize.

Version: 1.0	**Date:** 10/2/2019	**Priority:** 2

3.2.7 Base Unit 3D Dense Point Cloud

FR-3.2.7	Base Unit 3D Dense Point Cloud
Purpose: Interface for the Team coordinator to view a dense point cloud generated from the surroundings of the firefighter.	
Desc: This will allow the operator to view the physical surroundings of the firefighter with a 3D dense point cloud, with color. The operator will be able to not only see obstacles, but with the coloring, be able to identify what those obstacles are. Path of the camera is not included in this view.	

Input: Launching the base unit		
Output: GUI displays a 3D, dense, colored point cloud		
Error Handling: If the messages aren't being received' the application will still launch, but will say that no key frames are being received in the video stream. If the camera is moved to quickly, it will lose track of its frame of reference and output that its off track and trying to reinitialize.		
Version: 1.0	**Date:** 10/2/2019	**Priority:** 3

3.2.8 Remote Unit Settings

FR-3.2.8	Remote Unit Settings
Purpose: Options to configure the various options on the remote unit.	
Desc: When the "Settings" button on the remote unit's main menu is selected, a view for the modification of the remote unit's camera configuration settings will be displayed. These parameters will be for fps, resolution change, ip address, and topic names.	
Input: Configuration tab on remote unit is selected, various options can be changed from here	
Output: Configuration menu that has options for fps, resolution change, ip address, and topic names.	
Error Handling: If parameters are modified to be out of scope, they will be changed back to the default parameters.	

Version: 1.0	**Date:** 10/2/2019	**Priority:** 2

3.2.9 Base Unit Settings

FR-3.2.9	Base Unit Settings
Purpose: Options to configure the various options on the remote unit.	
Desc: When the "Settings" button on the base unit's main menu is selected, a view for the modification of the base unit's camera configuration settings will be displayed. These parameters will be for fps, resolution change, ip address, and topic names.	
Input: Settings tab on the base unit is selected, various options can be changed from here.	
Output: Configuration menu that has options for fps, resolution change, ip address, and topic names.	
Error Handling: If parameters modified to be out of scope, they will be changed to default parameters.	

Version: 3.0	**Date:** 10/17/2019	**Priority:** 2

3.3 Non-Functional Requirements

This section will explain the constraints and performance requirements in a quantifiable, and testable way. The non-functional requirements discussed are performance, reliability, availability, security, maintainability, portability, and scalability.

3.3.1 Performance

There will be three performance requirements for this distributed FIN system.

1. The first is the time it takes to send a ROS message from the node on the remote unit to the node on the base unit. After testing various messaging between units, all of the messages have been sent in near real time, all under 200 mill-second. This is the goal for the project for the final version. All ROS messages sent between nodes will be sent in under one second. This is after both applications have been launched; tests can be done by moving the camera and testing the time between the move and when the new frame appeared on the screen

2. The second is the boot time, or elapsed time after launching the executables on both nodes until the nodes are communicating and sending data to and from both units. It was agreed with our client for the boot time to be under 20 seconds.

3. The final performance requirement will be regarding the delay of the creation of the point cloud on the UI with respect to the live camera feed. The time it takes the process the ROS messages into a point cloud is greater than the time it takes to process those same messages into a live camera feed. Based on testing bag file to the video stream conversion on the base station laptop and ORB SLAM-2 documentation, this will be under one second.

3.3.2 Reliability

The reliability of the system as a whole is dependant on its hardware. Per **section 2.5** of this document, the working hardware assumptions state: the remote unit and base unit, screen, the depth finding camera, up board, the connection between the two, and the connected Wi-Fi module on the board are all functioning properly. Suboptimal bandwidth connections will not support a practical frame rate for the system and a wired connection will be used for proof of concept. For normal functionality the proximity of the

remote unit and base unit will also be a variable in the system's reliability; at most it can only handle about 30 feet between the two units before the connection is lost. From a software perspective, as long as these assumptions are met, the remote unit and base unit will reach a reliability standard. Possible software failures include the following:

- Failure to execute nodes on either unit
- Failure to convert video data to ROS image message
- Failure of node to publish/subscribe messages to topics
- Failure to process ROS bag to perform SLAM
- Failure to output video stream or point cloud to GUI

The main concern for program failures lies with the ROS network connectivity. The measurement we will use to test failure is MTBF (Mean Time Between Failures). Our requirement is to have above a 30 minute MTBF. If the MTBF is under 30 minutes, tests will be done to diagnose the purpose of the failure frequency and increasing the MTBF will be a high priority.

3.3.3 Availability

As there should be little need for maintenance on the master version after all functionalities are built and tested, the availability will be > 98%. The third party software/libraries used to facilitate the execution of the FIN system are saved locally on each unit and no change is expected. This is a closed system program, and relies on no state changing third party resources. The 2% time of unavailability, is an estimate for minor bug fixes if and when they appear.

3.3.4 Security

In the scope of this project, security is not a concern, nor a requirement. The client is not interested in security vulnerabilities in this semester. When the functionalities of this legacy project are extended in the future, security will be more of a focus.

3.3.5 Maintainability

Maintainability is a very important requirement for this product. Features will be added to the project after this semester. From a documentation, code structure, and understandability standpoint, this project must be clear and concise, in order to provide a new team the proper information to continue to expand functionality. The maintainability will be corrective, preventive, and perfective, but not adaptive.

One of the maintainability focuses will be corrective. Any bugs that are found during the testing process will be resolved before the final version is submitted. This will run parallel with the preventative maintainability. If the team agrees there is a likely chanced a similar bug could exist in further development, it will be well documented on what the bug was, how it was found, and how it was resolved. This is a preventative measure to be sure future development runs seamlessly.

This project was designed to specifically run with Ubuntu Linux 16.04, ORB-SLAM2, ROS kinetic, OpenCV 4.1.1. These technologies were agreed on between the client and the team and using different versions of these dependencies comes with no guarantee that the FIN system will function properly. This product will have no adaptive maintainability. Any changes in the build environment are not recommended.

As this is going to be a future legacy project, the existence of perfective maintenance is necessary. The team is well aware that stakeholders of this project will have new or changing requirements in the future, and the development will assure those requirements will be accommodated. The code will be well commented and structured intuitively to allow for straight-forward concept location upon the addition of new features.

3.3.6 Portability

Portability is a non-functional requirement of the FIN system, as it was designed and intended for a specific build environment. Running this program on a different version of Ubuntu, a different OS all together, or

with different dependencies/versions, will call for a major alteration of the codebase.

3.3.7 Scalability

A ROS network allows scores of nodes to communicate simultaneously, without creating a potential bottleneck. As new cameras are added to the distributed system, they are accompanied and paired with an additional up board, adding a greater processing load to the system overall, but not each individual remote unit. As all ROS image messages are routed to the main station and then processed with SLAM, this is where there is potential for bottlenecking. The processing power of the main console will be the determining factor of the maximum number of individual remote units. The connection the base station has with its network, and the Wi-Fi-transmission components in both units also have the potential to limit the scalability of the system, if the signal is suboptimal can there is only 15 megabits of bandwidth that will not allow the system to function with even 1 camera, even if all of the settings are turned down.

3.4 Design Constraints

There will be three constraints on the FIN distributed system. The range of the depth finding camera, the strength of the point-to-point Wi-Fi, and the processing power of the main station.

The Intel D435i camera has a range estimate with both a minimum and maximum. These values are dependant on the lighting conditions in the scene and camera calibration. The minimum range estimate is 0.11 meters and the maximum is 10 meters.

The strength of the point-to-point Wi-Fi will be dependent on the Wi-Fi module on both units, but particularly the remote unit, as well as the physical obstructions in the building (walls, floors, etc). At the time of version 2.0, the Wi-Fi module has been tested, and will not be sufficient for wireless communication between the units. Only about 1 frame per second can be output with the current hardware's bandwidth limits.

The processing power of the up board has been tested with the camera, processing video data and will not be a design constraint. The main console will have more of a workload, performing SLAM on the camera feed, and it's processing power is another constraint of the distributed FIN system. For details on the base unit and Wi-Fi module specifications and processing power (see **3.1.2 Hardware Interfaces).**

3.5 Logical Database Requirements

A traditional database such as those used in web development projects (SQL, MongoDB, etc.) is not required for this project. All data involved is either streamed live from the remote unit (video capture, depth data, IMU data) or generated in real-time on the base unit (point-cloud data and camera position data). Although no data will be sent to a remote server, individual keyframes from the video feed may be saved locally on the base unit. This data will be saved in a standard format of .png.

4 Analysis Models

This section contains several diagrams that help illustrate the concepts covered in this document. It outlines how the data is moving in the system and where it is being processed. It can be used as a reference to better understand the system's requirements.

4.1 Data Flow Diagrams (DFD)

There will be three data flow diagrams. The first will examine the distributed FIN system as a whole. The second and third will be taking a deeper look into each individual unit and how the data flow and is manipulated. In the first two diagrams, the start-point of the flow of data will be symbolized with a picture of the Intel depth finding camera.

4.1.1 Overall FIN Data Flow Model

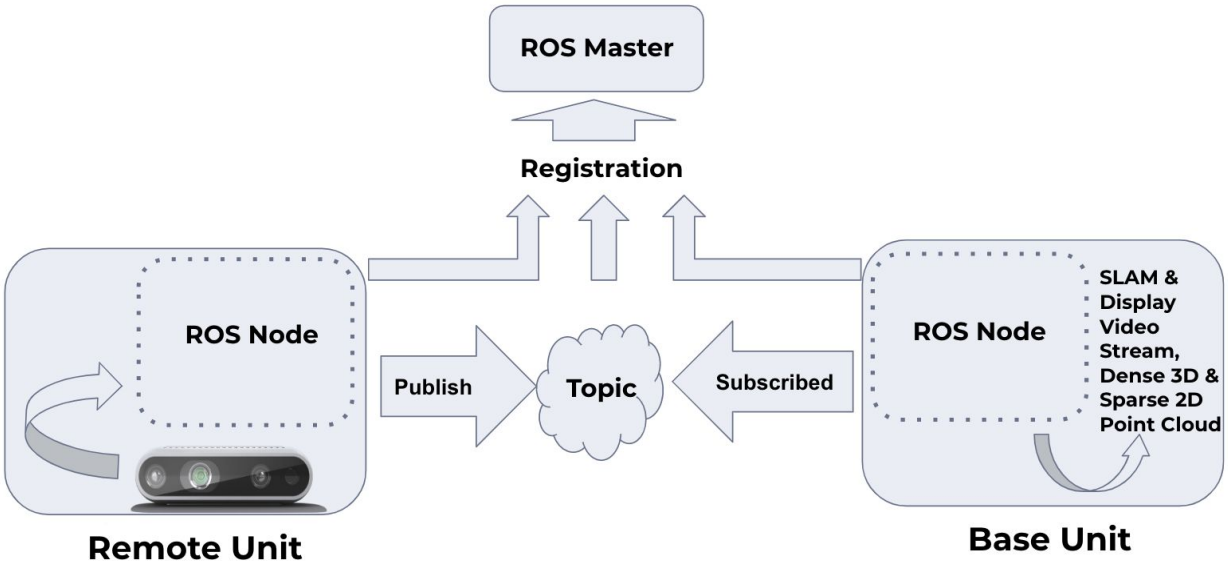

Figure 7

This diagram depicts both the remote unit and the base unit. A ROS node is an executable that can communicate with other nodes. Each unit will run a node for publishing/subscribing. A ROS message is the format in which data is sent through a ROS network. A topic is a bus that allows the flow of ROS messages from one node to another. Each node can subscribe or publish to a topic depending on the desired direction of the data flow. Each node and topic will be registered with the ROS master, which keeps track of which nodes are published or subscribed to which topics. This enables nodes to locate each other. In the remote unit, data received from the depth finding camera will be converted to ROS image messages.

Starting from the camera, the remote unit will receive the video data, convert it to ROS messages, publish it to a ROS topic. Since a node in the base unit is subscribed to that topic, it will receive those messages. This data transfer is represented the arrows interacting with the ROS topic cloud in figure 7. In the base unit, the newly received messages are processed. It is at this point that the point cloud maps and live video feed will be displayed on the base unit.

4.1.2 Remote Unit Data Flow Model

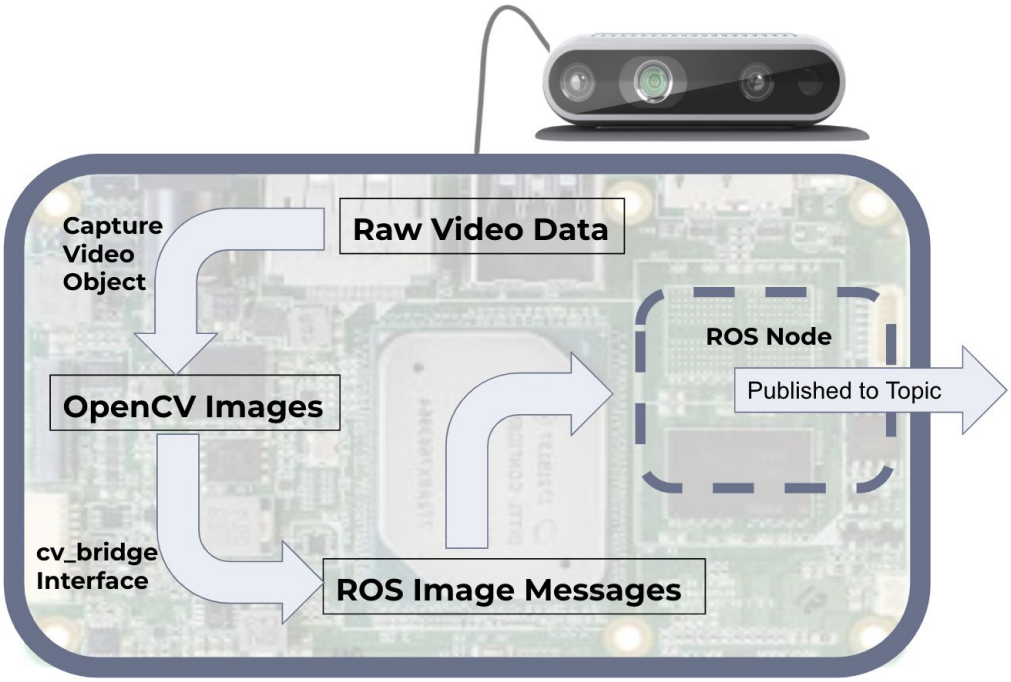

Figure 8

The remote unit of the FIN system facilitates the depth finding camera receiving data; it is received as raw video data. The camera is connected to the UP board via USB. There will be a ROS node running on this unit. It publishes ROS messages to a topic to send them to the base unit for SLAM .

Figure 8 displays how the FIN system converts the raw video data to ROS image messages in order for them to be sent to the base unit. Two classes from OpenCV will be used in order to make this conversion. An object of class VideoCapture will process frames from the video data frame by frame and prime it as OpenCV images for the next step. A cv_bridge object acts as an interface to convert OpenCV images to ROS image messages. Once the frames have been successfully converted to ROS image messages, they will be sent sequentially using the top node running on the remote unit to a node running on the base unit.

4.1.3 Base Unit Data Flow Model

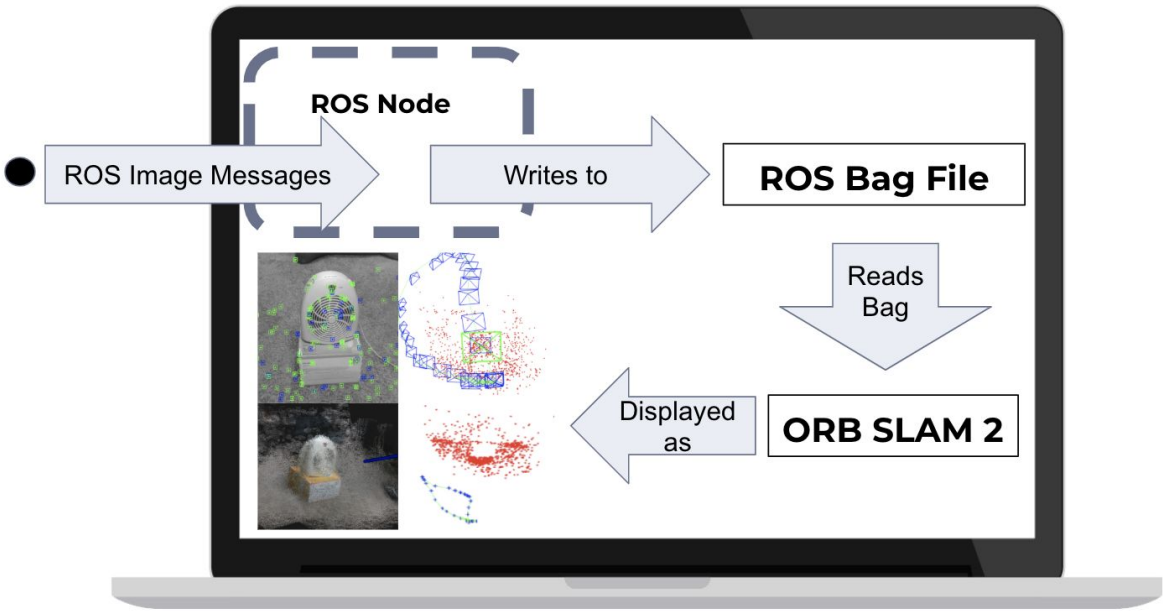

Figure 9

The base unit receives ROS image messages of video data, processes them with SLAM, and outputs multiple point cloud maps in its GUI.

In figure 9 data flow begins at the dot in the top left. The top node receives ROS image messages of video depth data which are then written to a ROS bag file, that bag file will be read simultaneously by ORB-SLAM2 to process that data and generate three point clouds. One point cloud will be sparse and 3D. Another will be sparse, 2D, and have a top down view for navigation. The third will be colored and dense for object recognition. There will be a video stream accompanying these point cloud maps.