Department of Electrical and computer Engineering

Part IV Research Project

Project Compendium

Project Number: 13

# Localisation and Mapping for Firefighters in GPS-denied Environments

Benjamin Yu

Jilada Eccleston

Dr. Kevin I-kai Wang

12/10/2018

# Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person outside the project team.

Names: Benjamin Yu & Jilada Eccleston

**ABSTRACT:** Currently, there are no commercially available mapping and localisation systems for firefighters. Even though prototypes have been developed to tackle this issue, not a single system has met the strict firefighting requirements. Such a system is critical for the safety of both firefighters and civilians. This research investigation involves reviewing the existing technologies, techniques, and systems, with intent to develop a mapping and localisation system for firefighters. The purpose of this project compendium report is to provide supporting material for information not included in the final reports and literature reviews conducted by the authors. It also serves as a guide for better handover of knowledge for researchers who intend to extend upon this study. Presented in this report is a low-level discussion of all the work conducted, a guide on how the system was set up, how to use the system, experimental procedure details, and documentation of the file structure within the compendium directory.

# 1. Introduction

### 1.1. Motivation

Firefighters rely on all their senses to navigate through unfamiliar buildings where floor plans are not available. In these environments, firefighters can easily become disoriented due to the heat, noise, and reduced visibility, which can lead to many fatalities. Firefighters are also required to describe the building's environment to their commanders after a smoke dive. Due to the stress that firefighters go through while inside the building, this map is often a very poor estimate of the ground truth. The birth of this study was driven by the interests of Defence Technology Agency around lidar technology, curiosity from the New Zealand Fire Service, and the need to improve the safety of firefighters and civilians. Before conducting a formal review of the literature, a brief investigation was done to understand the currently available systems. Surprisingly, there are no commercially available mapping and localisation system for firefighters available, hence, another motivation to further conduct research. The goals and scope of the project are clearly defined in the final report as well as the literature review and statement of research intent.

### 1.2. Report structure

This report is not stand-alone, it heavily references other files within the compendium. As stated previously, the purpose of this report is to provide supplementary information that was not included in the literature review and final report. It serves as documentation which contains information about the components within the research investigation. Therefore, enabling much easier handover of the project for future researchers who wish to extend upon this study. Section 2 presents a list of all compendium components, giving an indication of all the work done and an overview of the components within the compendium folder. Sections 3 and 4 present the system requirements and selection of sensors – the core components during the early stages of the study. Section 5 and 6 discuss the setup of hardware and software respectively. Section 7 explains the experimental methodology in more detail than the final reports. Finally, in Sections 8 and 9 future work and recommendations are briefly mentioned and conclusions are drawn. Note that the layout of this report has followed and been restricted to the provided template.

### 1.3. Work distribution

Due to the nature of this project, it was difficult to separate the workload. In the initial stages of the project the literature review, brainstorming, and planning were mostly undertaken in a collaborative manner. During the development stage, Jilada focussed on configuring the inertial measurement units, while Benjamin focussed on configuring the lidar, hence those sections have been written separately. The experimental, data collection, and analysis procedures were all worked on by both students.

## 2. Compendium Components

As per the requirements of the project compendium report, the following is a list of all hardware and software components of the project. It is essentially a breakdown of the work conducted in the project. Please note it does not completely reflect the compendium directory nor the structure of this report.

- **Project Compendium Report (this document)**
- **Course deliverables**
  - Literature Review and Statement of Research Intent reports
  - Final Reports
  - Poster
  - Seminar presentation
  - Project Video
- **Physical hardware**
  - Hardhat
  - Scanse Sweep lidar
  - Thunderboard Sense board x2
  - Raspberry Pi 3
  - Battery pack.
- **Software**
  - **General configuration**
    - Software environment setup
    - Networking setup
  - **MATLAB**
    - Error quantification
    - Firefighter motion
    - Xsens investigation
  - **ROS**
    - **Raspberry Pi**
      - Data collection and transmission
    - **Laptop**
      - Receive data
      - Laser scan matcher
      - Step detection
      - Fusion
      - OpenSLAM GMapping
  - **Embedded in Sensor**
    - Thunderboard sense board
    - Scanse Sweep lidar
- **Other documents**
  - **Hardware datasheets**
    - Scanse Sweep lidar
    - Thunderboard Sense
      - IMU-20648
  - **Experiment documents**
    - **Experimental Plans**
    - Photos of firefighter training facility visits
  - **Mechanical**
    - Raspberry Pi case
    - Thunderboard Sense board case
    - Lidar plate
    - Drawing of hardhat

## 3. System Requirements

The first component of this project was understanding firefighting procedures to define the requirements of the system. Under investigation in this study is indoor positioning systems. These systems contain several key components that distinguish indoor positioning systems (IPS) from global positioning systems (GPS). Signal scattering and attenuation plague the accessibility of GPS signals in density cities and indoor environments. Therefore, IPS have been developed to address these issues, enabling localisation in these environments. Firefighting is an example of a growing number of applications that have emerged requiring the use of localisation to improve safety and reduce risk. In addition to the obvious issues facing firefighters; harsh smoke-filled conditions, many other parameters need to be accounted for in the design of an indoor positioning system suitable for firefighting. Discussed in the literature reviews of both students are the requirements of these indoor positioning systems in the context of firefighting. For further discussion on these requirements, refer to those papers.

## 4. Sensor Technology Selection

The selection of sensors was attributed to the requirements previously discussed in the literature review. Due to the requirements of the system being catered towards both mapping and localisation, this opened a wide variety of sensor technologies to evaluate. Many sensors were compared, and a table was developed, which can be found in resulting in a table that can found in *other/sensor-selection/sensor-technologies.xlsx*. The tables presented in the literature review and final reports are condensed versions of this table. The information has been largely derived from the surveys conducted in [1] and [2].

After evaluating all sensors, it was concluded that the most feasible technologies were lidar, ultrasonic sensors, inertial measurement units (IMU), thermal imaging cameras, and radio communication via RFID tags or beacons. It should be noted that to ensure the evaluation was unbiased, no existing firefighting system was used to guide this analysis. However, to the relief of the authors, as shown in the literature review reports, many technologies in existing prototypes for firefighting mapping and localisation systems utilize these technologies.

It is envisioned that the final product will be a mixture of all five of these sensor technologies. However, for the initial prototype, it was decided to try lidar, ultrasonic sensors and IMU. Thermal and radio technologies are complex to setup given the firefighter application and one of the project requirements was to deliver a functional prototype that could be demonstrated to New Zealand Fire. Furthermore, the Defence Technology Agency had strong interests in understanding what the lidar could achieve for human navigation.

To decide on which sensors to purchase, a variety of sensors were evaluated. For a detailed evaluation please refer to *other/sensor-selection/product-selection.xlsx*. From the evaluation, it was decided to purchase the Scanse Sweep lidar, LV-MaxSonar-EZ1 ultrasonic sensors, and HC-SR04 ultrasonic sensors. A bill of materials of the final purchase can be found in *other/sensor-selection/bill-of-materials.xlsx*. The Thunderboard sense containing ICM-20648 was provided by the project supervisor.

During the early stages of the project, the team experimented with the newly purchased sensor technologies. However, it was discovered that ultrasonic sensors would be very challenging to work with. The sensors were one-dimensional, low range and limited resolution, in contrast to the lidar. Even though this was already known prior to purchasing the sensors, it was further realised that incorporating the data from such sensors and producing a pose estimate from one-dimensional data would be a complex task. For this reason, it was decided to focus only on lidar and inertial measurement unit technologies for the first prototype.

## 5. Hardware

As shown in Figure 1, the system is an interconnection of various sensors; LIDAR and IMU, to a Raspberry Pi 3 Model B that streams data wirelessly to the remote PC. To ensure that the connectivity between the sensors and the Raspberry Pi is stable, these sensors have a wired connection. The Raspberry Pi, LIDAR and IMU are all attached to a hardhat purchased from a hardware store). Plates have been 3D printed to easily fixate the LIDAR onto the top of the helmet. Holes have been drilled using a hole saw and an ordinary power drill. More details of the dimensions of these holes is shown in *other/mechanical/hardhat-drawing.pdf*. Other CAD design files can also be found in the *other/mechanical* directory. AutoCAD and CREO Parametric have been used to produce these files. Additional holes were drilled to

attach the Raspberry Pi case to the helmet. As the design considerations have already been covered in the final reports this section they will not be discussed here.


*Figure 1: Hardware overview*

The Thunderboard Sense was used for this investigation due to the wired Virtual COM connections and ease of prototyping. While there several different sensors on the Thunderboard Sense, only the IMU; ICM-20648 was used. While the Thunderboard Sense does have Bluetooth capabilities; which is known to be relatively stable, a wired connection for the IMU was still important to ensure that data losses do not occur. One of the IMUs is attached on top of the Raspberry Pi while the other is attached to the body of the firefighter. The specifications of both the Thunderboard Sense and the data sheet for the ICM-20648 are in the *other/datasheets* directory. The Scanse Sweep SEN-14117 lidar is a cost-efficient and long-range lidar. All specifications and operational procedures can be found in the datasheets folder. The Scanse Sweep lidar has no wireless capabilities and is connected through a USB connection straight to the Raspberry Pi.

## 6. Software

Software was developed for both the hardhat sub-system and the remote machine side. Software for the hardhat sub-system includes the development of the software for the sensors, namely the Thunderboard Sense which required changes to the example code provided to achieve the required sampling rate and wired serial data transfers. The Raspberry Pi also included software to perform serial transmission and pre-processing of data. Meanwhile, software developed on the remote machine side are mostly algorithms. The information flow presented in the final reports is shown in Figure 2.

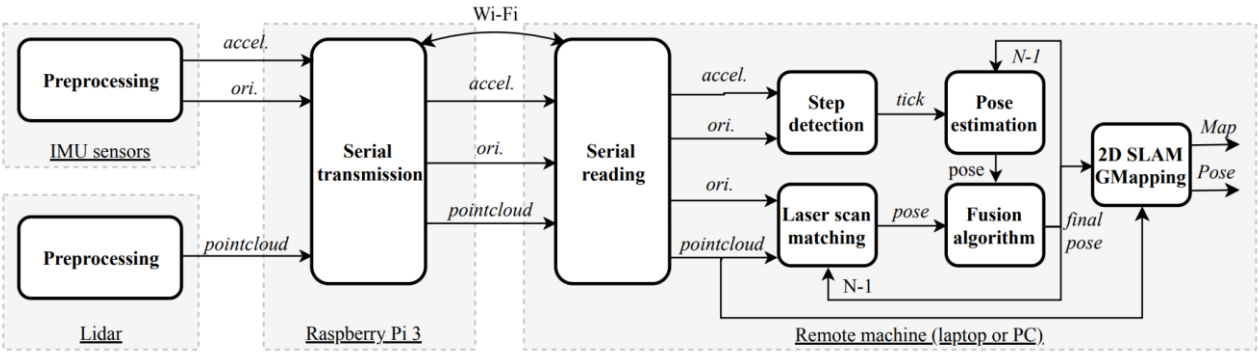


*Figure 2: Information flow*

### 6.1. Sensor software

Simplicity Studio was used to develop the program running on the IMU. To quickly prototype the system, the soc-thunderboard-sense example project was used. The soc-thunderboard-sense project contained sufficient code to connect,

run, and send data to the app using Bluetooth. The benefit of using this example project was the libraries were already integrated and implemented allowing the IMU data to be easily sent via serial to the Raspberry Pi.

Issues were encountered due to the system switching to low energy mode when Bluetooth was inactive for a sufficient amount of time. To remedy this, additional code was required to stop the device from going to sleep. Possible implications of deactivating low energy power mode can result in faster battery drain. However, the micro-B connection provides the device with power.

Additional changes needed to be made to increase the sampling rate to approximately 100Hz. Due to the bit operations that was occurring, it was not possible to achieve exactly 100Hz, so the frequency of sampling was approximately 102Hz. It was also noticed that by changing the *ACCELERATION_MEASUREMENT_PERIOD* and *ORIENTATION_MEASUREMENT_PERIOD* to lower than 10.5 resulted in the device sending data faster than the ICM-20648 can read new data; that is that data was spuriously all zero.

No alterations were made to the built-in software for the Scanse Sweep lidar. As stated in the lidar's data sheet a correlation algorithm is used to perform time of flight measurements. A point cloud data is generated by associating each distance value with an angle via an optical encoder. All details are clearly explained in the datasheet.

### 6.2. Raspberry Pi 3

To sample all the serial data transmitted to the device, a Raspberry Pi 3 running Ubuntu Mate was used; the installation steps are in "https://ubuntu-mate.org/raspberry-pi/". The Raspberry Pi was selected due to simplicity in prototyping. Additionally, the Raspberry Pi can run ROS a critical component of our sensor integration. Although the Raspberry Pi has several limitations, the most critical being the processing power. Wireless communication was set up to shift the processing of the data to another computer capable of running offline simulations as well.

Currently, the Raspberry Pi communicates to the remote machine by being on the same local area network. Two methods were used over the course of this project. During the beginning when only laboratory experiments were conducted, the Raspberry Pi and remote machine simply both connected to a wireless router in the laboratory. Another method involved setting up the Raspberry Pi as a router to provide a stand-alone local area network, the remote machine then connected to this network. The instructions to do this can be found in "https://jacobsalmela.com/2014/05/19/raspberry-pi-and-routing-turning-a-pi-into-a-router/".The purpose of the second method was so any experiments conducted in the field (outside the laboratory), such as in the training facility, did not require additional routers and could make a direct Wi-Fi connection from Raspberry Pi to the remote laptop. Further details on how we utilise this connection is detailed in Section 6.3.1.

### 6.3. ROS

The system utilises the open source Robot Operating System (ROS) framework, a set of well-documented software libraries, tools, and algorithms for robotic applications ("http://wiki.ros.org/ROS/Introduction"). ROS provides the means of not only ease of interfacing between different sensor parameters, but also access to industry standard libraries for sensors typically found in robotic applications. In ROS, several packages were created, spanning from serial sampling to pose estimation algorithms. Below, the internals of these packages will be explored in further detail. ROS advertisers and subscribers were used to provide connections between broadcasted topics. For convenience, launch files were used to instantiate instances of the packages and provide the associated parameters. It should be noted that it is necessary and highly recommended to work through the ROS tutorials found in "http://wiki.ros.org/ROS/Tutorials" before proceeding with any development in the ROS environment. The following will be presented assuming the reader has worked through the tutorials, this includes understanding concepts such as frames, packages, messages, topics, nodes and launch files. C++ was chosen as the core language of the system as both authors were familiar with that environment and all existing packages used in this project were written in C++.

For robotic-like systems operating in the spatial domain, it is important to understand the relationships between the many coordinate frames that might be used in the system. Figure 3 shows the tf tree, which essentially shows the relationships between the map, odom, base_link, imu_link and laser_frame coordinate frames. The purpose of displaying this information is because many of the errors during development are because the frames are not properly configured. For example, when configuring GMapping and laser scan matching packages, the relevant frames need to be specified correctly for the system to function as intended. For more information regarding the generation of tf trees please see "http://wiki.ros.org/tf".

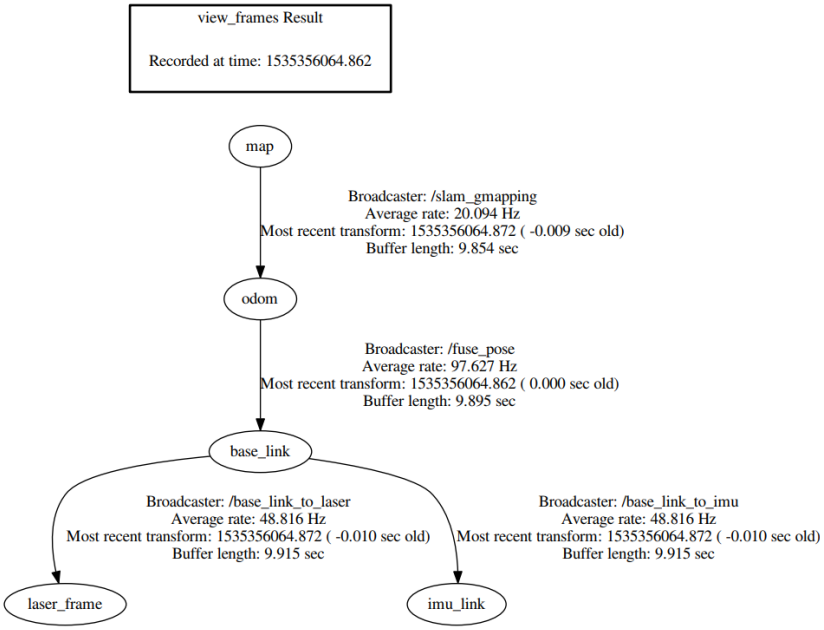


*Figure 3: Coordinate frame relationship*

### *6.3.1. ROS Setup*

The latest version of ROS; Kinetic, was used in the project. Kinetic was set up on both the Raspberry Pi and the Remote PC running Ubuntu 16.04. ROS was installed using instructions from the official ROS installation page for Raspberry Pi ("http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi"). The following details the setup process involved as well as the packages required commands are provided are for ROS kinetic - some variations can be observed when different versions of ROS such as Groovy are used. It is recommended that Kinetic is installed.

Wireless communication was required as during firefighter operations it is not practical to have a wired connection all the way to the commanding officer outside the building. Hence, to allow for online processing, so that the officer can receive the information as soon as possible, wireless communication was necessary. Note that it is still useful to perform offline processing and extract the data from the Raspberry Pi once the firefighter emerges from the building. This is because the information can still be used to help the next pair of firefighters ready to enter the building. The tutorial in "http://wiki.ros.org/ROS/Tutorials/MultipleMachines" was used to enable communication between multiple machines (the Raspberry Pi and remote laptop) within the ROS environment. Note that a requirement is that the machines are to be on the same network, as done in Section 6.2. This setup creates a ROS master and configures listeners and talkers to allow the two machines to share the same ROS environment. This resulted in published ROS topics that could be subscribed to from both machines.

As the Raspberry Pi by itself does not include a real-time clock (RTC), it is unable to accurately keep track of time unless it is connected to the internet. Each message in ROS has a timestamp associated with it. For the system to function it was discovered that the times on the two machines had to match and be synchronised. If it was out of synchronisation, then the data received on the remote machine could not be plotted in visualisation tools such as RViz (one of the default data visualisation tools in the ROS environment). Currently, an ad-hoc solution has been implemented where it is required to SSH into the Raspberry Pi every time it boots up. The instructions for this are in Section 7.1.2. A more elegant solution that is more time-consuming to set up is using an NTP time server, such as from the discussion in "https://askubuntu.com/questions/488072/setting-up-a-standalone-ntp-server-on-ubuntu". Due to time constraints, this has yet to be implemented.

After setting up the ROS environment various packages were installed. For this system, both existing ROS packages, as well as packages created from scratch were used. As discussed in "https://answers.ros.org/question/225538/how-to-install-ros-packages/", there are multiple ways to install existing ROS packages. The easier and preferred method is to

install the package straight into the system (added to PATH) such that all other projects can access the package. However, if one intends to edit the source code of an existing package it is easier to manage by cloning the repository into the working directory (or ROS workspace). Key packages that were used in the system will now be discussed in further detail, this includes both existing and self-built packages. Existing packages include sweep_ros, laser_scan_matcher, and GMapping. Self-built packages include imu_sample, imu_filter, and sensor_fusion. These packages can be found in the *ros_ws/src* directory.

#### *6.3.2. sweep_ros*

Scanse (lidar manufacturer) provides a data visualisation interface called Sweep Visualizer ("http://scanse.io") that allows configuration of lidar parameters such as speed and sampling frequency. The tool also allows saving the information in CSV file format. However, as this system required interfacing with ROS, this tool was only used to learn about the lidar and run initial performance tests, such as finding the actual range of the lidar.

Interfacing ROS with the Scanse Sweep Lidar involved installing the Sweep SDK provided by Scanse. To use the functions within the SDK, the tutorial in "https://github.com/scanse/sweep-ros" was followed, which resulted in point cloud messages being published to ROS topics. The existing code allows serial communication with the device, controlling parameters such as sampling rate, rotation speed, and reading data in. This code is in the file */ros_ws/src/sweep-ros/src/node.cpp*. An additional package called pointcloud_to_laserscan ("http://wiki.ros.org/pointcloud_to_laserscan") was also installed to convert the point cloud messages output from sweep_ros to laserscan messages which is accepted by the laser_scan_matcher package.

#### *6.3.3. laser_scan_matcher*

An existing package called laser_scan_matcher ("http://wiki.ros.org/laser_scan_matcher") was used to calculate a pose estimate from the laserscan data from the lidar. To the team's knowledge, this was the only algorithm available that performed laser scan matching without the need of robot odometry. Note this laser_scan_matcher is only a package within a module called scan_tools ("https://github.com/ccny-ros-pkg/scan_tools"), it is located in *ros_ws/src/scan_tools/laser_scan_matcher*. The package employs an iterative-closest-point (ICP) matching algorithm that finds the optimal transformation between subsequent laser scans that minimises the difference between them. The transformation is then applied to the previous pose (position and orientation) estimate to provide an estimate of the next pose.

Initially, the laser_scan_matcher package was simply installed through the normal ROS method of installing packages and the internals were treated as a black box. However, after running preliminary tests with the package, the scan matching would produce internal errors. In other words, the package is not robust; the same issue is shown in "https://github.com/ccny-ros-pkg/scan_tools/issues/20". Because of this, to install this package, the repository was cloned into the working directory so that edits to the source code could be made more easily. After analysing the internals of the package, it was found that the algorithm was exhibiting convergence issues when the input laser scans were too different from one another (due to fast movements). To solve the issue of convergence, a few additional lines of code were added to provide a better initial guess that took rotation information from the inertial measurement unit on the helmet. Furthermore, changes were also made to allow the laser scan matcher ROS node to subscribe to the final pose estimation from the fusion stage. This is essentially a feedback mechanism to replace the previous pose estimate with the fused pose estimate, which should be more accurate.

#### *6.3.4. GMapping*

The GMapping ROS package ("http://wiki.ros.org/gmapping") is only a ROS wrapper for OpenSLAM's Gmapping ("https://openslam-org.github.io/gmapping.html"). The package performs laser-based simultaneous localisation and mapping to create a 2D occupancy grid map. The inputs parameters to the package are pose and laser scan messages. The output parameter is an occupancy grid message in ROS. The internals of the algorithm were treated as a black-box.

Another package called hector mapping ("http://wiki.ros.org/hector_mapping") was also explored as its description was ideal for our system. It was designed around systems that did not have wheel odometry data. However, it requires lidars with very high update rates. Early experiments did not produce clean maps as the Scanse Sweep has a relatively slow update rate, in contrast to the Hokuyo lidars the algorithm was designed around. During the early stages of the project, the team quickly switched to GMapping as it was easier to setup and configure. Because of this, the package could be investigated in the future by conducting a more in-depth comparison.

#### *6.3.5. imu_sample*

Both *imu_sample_node* and imu_publish_node belong to the imu_sample package which was created to sample the serially transmitted IMU data. The data transmitted will then be either processed and stored into a CSV file for further processing in MATLAB (imu_sample_node) or converted into an Imu_msg type to be transmitted over rostopics (imu_publish_node). As the code is written in C++, additional files are needed to allow for serial communication to occur ("https://github.com/fedetft/serial-port").

The program is multithreaded utilising the Boost libraries and can be scaled to introduce more hardware sensors. Changes will need to be made in the data that is being sent by introducing further identifiers for the different sensors - currently, the data is appended with an identifier unique to each individual IMU sensor.

In order to publish the node some of the parameters supplied will need to be converted into the form accepted by Imu_msg; a ROS message belonging to sensor_msgs ("http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html"). It is important to also consider that the *CMakeLists.txt* file requires some adaptation to compile the nodes due to the use of Boost libraries. Notably, *add_compile_options(-std=c++11)* which changes the C++ compiler to a newer version to compile some of the more recent Boost library files.

Before developing the algorithms, initial visualisation of the data was performed to understand the data and operation of inertial measurement units. Rqt_plot ("http://wiki.ros.org/rqt_plot") was used to graph the data, while rviz_imu_plugin ("http://wiki.ros.org/rviz_imu_plugin") was used to visualise the orientation in 3D.

#### *6.3.6. imu_filter*

Step detection mitigates the exponentially accrued error that is encountered during double integration. Instead, the step detection processes acceleration data to determine when the step has occurred and accumulate distance in discrete time steps as opposed to a more time continuous method that is double integration. Using this method, the previously exponentially accrued error is reduced to a linearly accrued error.

As ROS is primarily used for robotic systems, there does not exist any existing packages around step detection. For this reason, an algorithm has been built from scratch. All the code for the step detection algorithm can be found in the imu_filter package. This package contains a custom message type; parse.msg. The Header type is a standard ROS message type ("http://docs.ros.org/kinetic/api/std_msgs/html/msg/Header.html").

Parse:

- Header header
- float64 variance
- float64 average
- float64 diff_squared

The variance, average and diff_squared variables all relate to the algorithm that processes them. This was experimented with to determine if the signal reacted better to different types of pre-processing before extracting the step information out.

IMUStore is a data structure that holds information about data received from individual IMUs. Each IMU will have its own IMU_Store. During instantiation, the constructor requires a *maxWindowSize* which is used to determine the size of the vectors storing the orientation (as a quaternion) and acceleration. The *updateWindow* performs the processing of the data stored in the vector, resulting in the variance for the X, Y and Z accelerations to be determined. The variance values are then subsequently used in the *stepDetectionVariance* function to determine if a step has occurred. Currently, the function only considers the acceleration in the relative Y-axis. Following this order, the newest packet of data can be retrieved. The broadcasting of the variance estimations contributes only to the visualisation of the data.

IMUPosition processes the latest position estimate if a step has occurred using the most recent orientation measurements. Constructing the IMUPosition class results in the initialisation of variables including the step length used for accumulating the position estimates. This has been set to 0.5m, however, better relationships can be used that combine the measurements of the target and acceleration currently experienced to determine a more accurate step length.

Yaw measurements for both the IMUs are continually updated. Once the step is detected, *calculateNewLocation* resulting in the calculation of the new current X and Y coordinates. Two measurements of the position can be estimated, the first continuing from the previously estimated IMU only pose and the second continuing from the fused pose fed back into the algorithm. The ultimate goal of the IMUPosition class is to return a ROS point type that is part of geometry_msgs ("http://docs.ros.org/kinetic/api/geometry_msgs/html/msg/Point.html").

To bind both the IMU data storage and the position processing, imu_filter_node is used. The imu_filter_node contains subscribers to the raw data sent by both the IMUs. Publishers are used for broadcasting the filtered data and the resulting position estimate from the processing. Callback functions *ImuCallbackBack* and *ImuCallbackHelmet* process information received from the imu_data_A and imu_data_B rostopics. Currently, the position estimate is being gathered from the IMU attached to the helmet. Data is stored in the IMUStore and then processed in IMUPosition. To begin processing the estimate, an average is required to be calculated, therefore requiring a sufficient number of data points collected. *MAX_WINDOWSIZE* defines the size of this window and is both the input to the IMUStore used to prevent early processing.

A rest period defined by *REST_PERIOD* is used to ensure that steps can only be detected after a reasonable time. For experiments, this has been sent to 10 points. Helper functions are defined at the bottom of the file and serve to isolate areas of critical quaternion operation. These functions include the creation of the PoseStamped data type ("http://docs.ros.org/kinetic/api/geometry_msgs/html/msg/PoseStamped.html") and the quaternion matrix operations to update the current delta rotation ("http://docs.ros.org/kinetic/api/tf/html/c++/classtf_1_1Quaternion.html").

The new PoseStamped measurement is generated when a new step is detected in either of the helmet or the back. This combined measurement allows us to mitigate the effects of unaccounted steps by any of the motions conducted by the firefighter. Additionally, a boolean is determined which specifies whether the head is tilted or not. This boolean is determined using the orientation measurements retrieved from the IMU attached to the helmet in *ImuCallbackHelmet*.

#### *6.3.7. sensor_fusion*

The intention of this package is to provide the best possible pose estimate to then be fed into the GMapping algorithm. The node subscribes to the pose estimates from the step detection and laser scan matcher algorithms and publishes a pose estimate of its own. It also subscribes to a topic called lidar_valid which is currently output by the imu_filter node. The current implementation is a simple technique. When the lidar is invalid, the final pose estimate is purely dependent on the step detection algorithm. Similarly, when the lidar is valid, the final pose estimation is purely dependent on the laser scan matching algorithm. This final pose estimate is also fed back into the step detection and laser scan matching algorithms to replace their previous estimate.

### 6.4. Launch files

Now that all packages used in the system have been discussed, this section will now present how the packages are used and launch files are used to run the packages. A launch file is a way of reducing development time by providing a convenient way to start up ROS nodes. Each launch file can start multiple nodes and configure parameters for each node. These launch files have been developed within their own package, called slam_launch. The files can be found in *ros_ws/src/slam_launch/launch*

#### *6.4.1. pi.launch*

This launch file must run on the Raspberry Pi as it runs the nodes necessary to interface with the hardware sensors attached to the Raspberry Pi itself.

#### *6.4.2. record.launch*

As the nodes launched by pi.launch publish ROS messages to ROS topics this data must be stored into rosbag files (a file type used to store ROS messages) to allow for offline processing. This file simply stores the raw data from the sensors, the imu_data_A, imu_data_B, and pc2 ROS topics. These topics correspond to the linear acceleration and orientation data from the inertial measurement units and the point cloud data from the lidar.

#### *6.4.3. pc.launch*

This launch file runs on the remote laptop or PC and is used for online processing. It establishes the necessary transforms between coordinate frames, begins the pointcloud_to_laserscan node, starts the fusion node, starts Gmapping, begins RViz for mapping and localisation visualisation, and finally runs additional rqt_plot nodes to visualise inertial measurement unit data in the form of time series graphs. Parameters configured for each package is inside the launch file if required.

#### *6.4.4. pc_offline.launch*

This launch file is mostly a replica of pc.launch, except it runs on offline data. It does this by playing back the rosbag file previously recorded in the online session from pi.launch. One important point to note is that it uses the simulation time, not the current ROS time.

### 6.5. MATLAB

A few MATLAB scripts were written throughout the duration of the project purely for the sake of quick processing of data. These scripts are in the *other/matlab* folder in the compendium directory. The error-quantification is the script used to calculate the localisation error during the straight-line tests. The data was converted from rosbags to CSVs and then processed by the script. In the *firefighter-motion* folder was an attempt to analyse the data in MATLAB rather than C++. The purpose of the script was an attempt to find patterns in the linear acceleration data by plotting metrics such as energies and sums. The same method was performed in [3], where pedestrian movements were able to be classified. However, the IMUs were not attached onto the foot like in the study and the results could not be reproduced. Finally, in the *xsens-investigation* folder includes scripts that were used to perform double integration on laboratory data collected from Xsens MTi units. The conclusion from that investigation was that double integration was not feasible for this application due to large amounts of drift.

## 7. Experimental Methodology

Experiments were initially conducted in the Embedded Laboratory in Newmarket Campus. These experiments include the investigation into the integration of the IMU and LIDAR in ROS. These experiments focused on the fine-tuning the parameters for the algorithms in GMapping and laser_scan_matcher. Eventually, the system was stable enough to test at the firefighting training facility at the Takapuna Fire Station. Lab experiments and discussion of experimental results are already discussed in the final reports of both the authors, hence are not presented in this report. Full experimental plans are available in the *other/experiments* folder. The initial procedures for both the experiments include a debriefing of the firefighting personnel. All recorded data in the form of rosbags is in *ros_ws/src/rosbags.*

In the laboratory, a desktop computer (native Windows 10) running Ubuntu Linux 16.04 Desktop in VMware Workstation 12 Player was used to run the data processing algorithms. 2GB of RAM, 25GB of storage and 2 cores were allocated to this Virtual machine. For the experiments conducted at the training facility, a MacBook Pro 2017 was running Ubuntu Linux 16.04 Desktop in a Parallels Desktop 12 Virtual Machine. 2GB of RAM, 64GB of storage and 2 cores were allocated to this Virtual Machine.

### 7.1. System Usage

The following describes the steps taken to use the system.

#### *7.1.1. Calibration*

Calibration of the system includes resetting the IMU it remained relatively stable. The result of not allowing the IMU to be stable while the calibration occurs will result in fast drifting of orientation data. To mitigate these effects, the attached of the power bank is used to signal the calibration process as all the devices turn on. Vibrations due to the rotation of the LIDAR when powered up will naturally result in high drift in the IMU calibration. It is advised that the calibration occurs in such steps:

1. Set the prototype on a flat surface
2. Unplug the LIDAR from the Raspberry Pi
3. Ensure the power cable and the USB cables attached to the IMUs are connected
4. Connect the power bank and wait until the red LEDs on the Thunderboard Sense appear - these signal that the calibration and initialisation process has completed.
5. Connect the LIDAR to the Raspberry Pi

#### *7.1.2. Software instructions*

The Raspberry Pi credentials are p4p (username) and password (password). For networking, IP addresses may change, this may require edits to /etc/hosts file, as per the tutorial for allowing communication between multiple machines in ROS. Similarly it may be required to change the ROS master, depending on what network the systems are connected to, this can be done in the .bashrc file, as per the same tutorial. Assuming the ROS environment, networking, and ROS packages are correctly configured, the following details the steps that should be undertaken to use the system in an online fashion To use the system in an offline fashion, simply run pc_offline.launch without the below launch files to avoid conflicts.

Terminal 1

- SSH into the Raspberry Pi by:
    - ssh p4p@192.168.8.1 (IP address may be different)
    - password
- Manually change the date of Raspberry Pi by grabbing the date from the laptop by:
    - sudo date -s "$(ssh LAPTOP_USER@IP_ADDRESS_OF_LAPTOP date)"
- Start pi.launch by:
    - roslaunch slam_launch pi.launch

Terminal 2

- Start recording by:
    - roslaunch slam_launch record.launch

Terminal 3

- Start pc.launch by:
    - roslaunch slam_launch pc.launch

### 7.2. Firefighter Training Facility

Firefighters; both career and volunteer, train at the Takapuna firefighter training facility every Monday. Three experiments were conducted at the firefighter training facility. These involved fixing the prototypes onto a firefighter conducting a training exercise and recording the results. The final two tests conducted involved both the mapping and localisation aspects of the prototype. Equipment used in this experiment includes the prototype itself. A secondary power bank was brought to the training facility in case the primary power bank runs too low. However, a single fully charged power bank will suffice for the entire experiment, even when left on during the entire duration.

#### *7.2.1. First Visit*

The first set of data collection with firefighters was conducted with the Xsens MTi IMU where a wired connection was not available. The Xsens MTi IMU while accurate faces massive connectivity issues when there is no line of sight to the USB connector (this may have been due to the old units given to the team). Through a proprietary protocol, Atwinda, data can be buffered and transmitted to the base station. During the test, the IMU was consistently lost connection with the remote PC which meant that data could not be reliably recorded and processed. Soon after, it was decided that a different more robust sensor is explored. The experienced allowed the group to understand the practical issues around data collection and resulted in the team requiring a better experimental procedure.

#### *7.2.2. Second Visit*

This was conducted on the 27th of August 2018. The second visit started with explaining the procedure to the firefighters. Including providing some background information into the system designed. The technologies were also briefly explained.

During the briefing of the firefighters, a test run of the system was conducted to ensure that connectivity could be established, and the system was somewhat calibrated and stable. This was not worn by the firefighters instead, by the author.
To ensure that the firefighters were not biased in any way, the first scenario involved instructing the firefighter to operation as per normal. This was conducted in full light conditions to capture the motion on camera. The footage for this can be found in *other/media*, titled *ftf_scenario0_normal.MOV*. The next scenario conducted was the firefighter instructed to not tilt their head and only perform the standard walking motion. As with the first scenario, the test was conducted in full light and footage for that can also be found in *other/media*, titled *ftf_scenario1_notilt_nostomp.MOV*.

The third scenario was conducted in the same fashion as the first scenario however the lights were switched off. Therefore, the motion of the firefighters could not be regulated. In this set of experiments, the effects of smoke were going to be tested however, due to the unfortunate injury of one of the firefighters, the exercises were called off and the session ended.

#### *7.2.3. Third Visit*

This was conducted on the 3rd of September. In the previous experiment, smoke data was not collected. In the third visit, there was a heavy focus on understanding the effects of smoke on the device. It was also noted that the firefighters operated using different motions. Four were selected to focus on:

1. Normal Walking - used as a ground truth
2. Kneeling and stomping with one foot
3. Walk and stomp
4. Walk and sweep

Footage of the four motions can be found at *other/media*, titled *ftf_motion_[motion type][X]*. Where motion type can be one of kneeling, stomping, sweeping, or walking, and X denotes the trial number. Two runs were done for every motion type and the footage for all runs have been recorded and labelled to reflect the rosbag file name.

While the experimental plans detail information on four scenarios investigated, due to the result in smoke, the other experiments could not be conducted. Scenario 1 was conducted where the building was filled with smoke generated from a dry ice machine. Zero-visibility is encountered like the visual environment exposed to firefighters in the scene. However, the heat intensity is not simulated.

## 8. Future Work and Recommendations

Many environmental challenges are faced in this investigation. These challenges range from connectivity to the reality of smoke-filled conditions. While in the smoke-filled conditions, the system could not function; a large and unavoidable limitation of the technology used. It is recommended if LIDAR is something that continues to be pursued, a better and more advanced LIDAR is used instead. However, this leads to the concern of firefighters as the cost of these technologies increase the less likely these systems will be adopted in the exercises. A prime example of this is the thermal imaging camera. Many aspects of the project can be further developed in more details. An elaborated explanation of the future work can be found in the final reports of both the authors.

For future development of this system it is recommended to identify the scope of the project. As this is an initial prototype and the first time running this project at the University of Auckland, it was natural to have a wider scope and experience many problems. In the future, research groups should focus on specific tasks, such as detecting firefighter motion, improving the mapping quality in different scenarios, or applying advanced signal processing techniques on the complex signals.

## 9. Conclusion

This report provides an in-depth view in to the research and development of a localisation and mapping system. Due to the interest in LIDARs from the sponsor, DTA, these sensors have been the focus of the investigation and incorporated in the final design. To complement the LIDAR in positioning, IMUs have also been investigated. While the system addresses several the concerns faced by firefighters, it fails to function during smoke conditions. Further work must be considered that advances the fusion methodology and the sensors used to balance the cost of the system to its capabilities. The motivation for this continuing research is compelling, where there is obvious market interest in a commercial product as no current existing system in place to tackle these issues.

## Acknowledgements

This was a project undertaken by Jilada Eccleston and Benjamin Yu from the Electrical and Computer Engineering Department, University of Auckland. The students would like to thank the project supervisor Kevin Wang, PhD candidates Andrew Chen and Qinglin Tian for their guidance and support. Special acknowledgements to the Defence Technology Agency for their sponsorship and Devonport Fire Brigade for their cooperation and permission to collect data with real firefighters. Furthermore, the fire brigade supplied information of firefighting procedures and existing systems that was difficult to procure through online resources. Finally, specially acknowledgements to the technicians and academic staff at the University of Auckland to make this project possible.

## References


[1] R. Mautz, “Indoor Positioning Technologies Habilitation Thesis submitted to ETH Zurich Application for Venia Legendi in Positioning and Engineering Geodesy,” ETH Zurich, 2012.

[2] J. Rantakokko *et al.*, “Accurate and reliable soldier and first responder indoor positioning: Multisensor systems and cooperative localization,” *IEEE Wirel. Commun.*, vol. 18, no. 2, pp. 10–18, 2011.

[3] M. Lee, C. Park, and C. Shim, “A Movement-Classification Algorithm for Pedestrian using Foot-Mounted IMU,” *Proc. 2012 Int. Tech. Meet. Inst. Navig.*, pp. 922–927, 2012.