

Visual telemetry transmission in marine environment using Robot Operating System platform

Barakou Stamatina
Chasapopoulos Konstantinos

Diploma Thesis



School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Professor G. Papalambrou
Committee Member: Professor K. Kyriakopoulos
Committee Member: Associate Professor C. Tzafestas

September 2019

Acknowledgments

This work has been carried out at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, under the supervision of Assistant Professor George Papalambrou.

We owe our greatest appreciation to our supervisor Assistant Professor George Papalambrou for giving us the chance to work on this thesis. In addition to his constant guidance, valuable comments and fruitful discussions, we have to thank him for his unceasing encouragement and support through this thesis.

We are also grateful to our family and friends for their constant encouragement throughout our thesis.

Abstract

The present thesis intends to research on the development of a telemetric communication system application between two distant locations. The purpose of this application is to secure an underwater area with the help of a floating base, which is supervised by a land-based control center. The floating base is unmanned and instead it is equipped with a group of sensors that gather information regarding its status and the state of the underwater area. Battery power, geographical location, temperature and pressure levels sensors ensure the autonomy and safety of the device. Finally, a motion camera which automatically detects human presence is placed on the floating base in order to secure the underwater area from possible trespass. Sensors' data is collected and transferred to the remote control center. With the use of an integrated application, a qualified operator has access to all the data listed, indicating in what state the floating base is as well as if there was a human trespass occurrence.

This description summarizes the project's basic idea. For the actual implementation, a small simulator was created in the Laboratory of Marine Engineering at the Technical University of Athens and consists of two machines. One PC is named Unit point and corresponds to the floating base and the other one is given the name Base point corresponding to the land-based control center. Each one of them represents one of the distant points.

Contents

Contents	4
1 Introduction	7
1.1 Problem Description	7
1.2 Literature Review	9
1.3 Thesis Structure	10
1.4 Contributions	11
2 Background	13
2.1 Robot Operating System	13
2.1.1 History	14
2.1.2 Why Use ROS	15
2.1.3 Applications Using ROS	16
2.1.4 ROS Terminology	17
2.1.5 Message Communication	22
2.1.6 ROS communication models differences	25
2.1.7 Analyze Message Communication Flow	26
2.1.8 Messages	31
2.1.9 ROS Tools	34
2.2 SSH Definition	41

2.2.1	Set up ssh keys	41
2.2.2	How it works	42
2.3	Database	43
2.3.1	Evolution of Databases	44
2.3.2	SQL Definition	46
2.4	OpenCv	49
2.4.1	OpenCv Definition	50
2.4.2	Computer Vision	50
2.4.3	Haar-Cascade Detection in OpenCv	53
2.4.4	YOLO	55
2.4.5	OpenCv in ROS	59
2.5	Hypertext Markup Language - HTML	59
2.5.1	HTML Definition	59
2.5.2	How HTML works	60
3	Implementation	68
3.1	ROS	68
3.1.1	ROS Communication models	68
3.1.2	UnitCall Service	70
3.1.3	BaseCall Service	73
3.1.4	ROS Master	73
3.1.5	Network Setup in ROS	74
3.1.6	OpenCV	75
3.1.7	Sql	80
3.1.8	Html	82
3.2	Application Description	86
3.2.1	UnitCall Development	88

CONTENTS **6**

3.2.2	BaseCall Development	93
4	Results	98
4.0.1	Application prerequisites	98
4.0.2	Application Startup	99
5	Conclusions and Future Work	105
5.0.1	Conclusions	105
5.0.2	Future Work	106

Chapter 1

Introduction

1.1 Problem Description

This work is part of a feasibility study carried out at the School of Naval Architecture and Marine Engineering/NTUA with main objective to investigate the capability of telemetry transmission between two distant locations in sea environment. In an abstract formulation, the problem at hand is to set up suitable software and hardware components so as to survey an underwater area (local station) and transmit various kinds of information in the form of “messages” to a distant control center (remote station). The local station has to be unmanned and equipped with a group of sensors that gather information regarding the state of the underwater area. In addition, choices for entities like suitable battery power, geographical location and communications must ensure the autonomy and safety of the station. The main feedback comes from an camera imaging system which allows a post-processing application to automatically detect presence of objects of any kind (ie divers, ROVs, fishing vessels etc). The chosen platform for the construction and handling of messages is the Robot Operating System (ROS) framework, well established in the academic as well as in the industrial environment. In addition, other mechanisms were set up and co-operated with ROS: a basic image processing application that processes video and pictures that come from the camera sensor initiating the messaging generation process, in the local station side, as well as a data base application

that has to collect and store the various types of messages that are received, together with a display environment for monitoring the whole remote operation by a human operator; both in the remote station side. Figure 1.1 shows the essential components of the complete application.

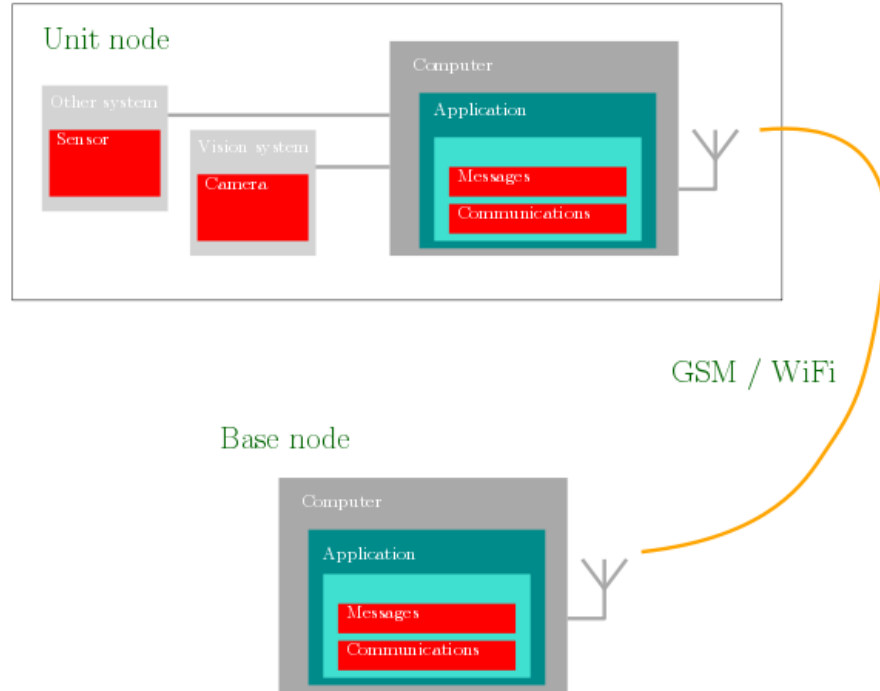


Figure 1.1: A National Technical University of Athens project, which aims to construct a floating surveillance system

As for the contents of this Thesis, a minimal simulator 1.2 was set up at the Laboratory of Marine Engineering/NTUA, consisting of two “computing nodes”: one laptop named Unit point which corresponds to the local station at sea, and a second laptop, given the name Base point, corresponding to the land-based control center. In between an Ethernet network was set up so as to simulate the link between the two nodes, with varying traffic and connectivity performance. Both computers run Linux operating system.

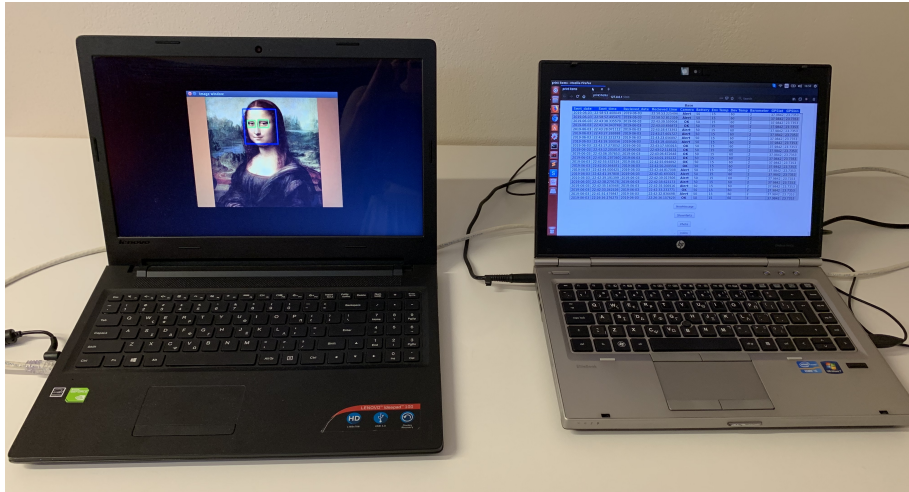


Figure 1.2: Simulator consisted of two machines connected via ethernet cable

1.2 Literature Review

The Robot Operating System (ROS) is a framework and set of tools that provide functionality of an operating system on a heterogeneous computer cluster. ROS applications have been implemented since 2007. ROS provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines, tools for testing and visualization, and much more. Its usefulness is not limited to robots, but the majority of tools provided are focused on working with peripheral hardware.

Many systems are using ROS communication not only for navigation and trajectory planning but also for telemetry and monitoring. Specifically, in [11], eye tracking technology is used for telemetry robot control. The telemetry system consists of a graphical user interface with reconfigurable multiple widgets whose transmission is prioritised based on the user's gaze. The proposed approach encompasses a novel framework for telemetry of remote machines, based on ROS. It includes a modular ensemble of GUI, gaze device interoperability, and a ROS sensory topic modulator, which alternates different strategies to optimise all

displayed information transmission quality.

In [12], a multi-agent structure based in ROS is implemented for autonomous surface vehicles. This work describes a multi-agent navigation guidance and control system that is designed to equipping an autonomous surface vehicle. In the developed of the project, a small autonomous surface vehicle is conceived to perform as component of an integrated platform, together with a micro remotely operated vehicle. The autonomous surface vehicle is able to recover the remotely operated vehicle, through a radio link, from a shore-ground station. The platform can be employed for exploration, monitoring and light intervention purposes in the underwater environment, allowing telepresence and direct guidance of the operator on the spot of interest without the costs of manned supply vessels. The autonomous surface vehicle contains sensors to acquire and publish, within the agency, position, attitude, axial rotational speeds and accelerations of the vehicle obtained by GPS and IMU. A surveillance camera acquires and streams video and manages tilt and focus according to external requests. Additionally, a logger is contained in order to save in a common file format all the data generated and exchanged by the agents. ROS helps to monitor the status of the infrastructure and to alert about the system's failures.

In [13], a ROS based stereo vision system is constructed for autonomous vehicle. The design contains an autonomous vehicle which is powered by ROS. The vehicle is capable of maintaining a constant speed and distance for monitoring or surveillance. ROS is implemented for trajectory tracking and telemetry. The vehicle is powered by a compact embedded system. Various image processing techniques are been implemented for navigation and obstacle detection. Artificial Neural Network is used for finding the shortest path by acquiring data from image processing.

1.3 Thesis Structure

In the following chapters there will be a thorough analysis of the tools that were used for the implementation of the application and an etiology of how their combination made the application operational.

This thesis combines different research fields in order to conduct an integrated

application. **Chapter 2** presents the essential components and concepts of ROS, as well as core parts of the application. Core parts include OpenCv that was used for image processing, Secure Shell (SSH) for the machines' safe connection, Structured Query Language (SQL) for data storing and HyperText Markup Language (HTML) for the application platform. **Chapter 3** describes the project's implementation. This includes the basic ROS communication design and its integration with the core parts. **Chapter 4** presents the results after running the integrated application on the small simulator that was created. **Chapter 5** presents the conclusions depending on the basic idea project and the result from the simulation. Also, some thoughts for future work are proposed as well.

1.4 Contributions

Barakou Stamatina and Chasapopoulos Konstantinos contributed to the development of the project and the final version of the writing. The floating base (Unit point) was undertaken by Stamatina and the land-based control center (Base point) was undertaken by Konstantinos. Both authors implemented the communication between Unit and Base. More specifically,

Stamatina Barakou:

- established the SSH Unit-Base connection on the Unit side
- implemented the client side communication using ROS software
- developed a face recognition system using OpenCv that simulates human detection
- designed a message management system where messages are sent automatically to Base

Konstantinos Chasapopoulos:

- established the SSH Unit-Base connection on the Base side
- implemented the server side communication using ROS software

- managed message extraction, classification and database storage
- designed the application platform

Chapter 2

Background

2.1 Robot Operating System

Nowadays, the power of Internet has given people great opportunities to expand their knowledge even if they are researchers and developers trying to open new scientific fields or even if they are students and their journey of knowledge just began.



Figure 2.1: Robot Operating System is an open source tool to control complex robotic systems [1]

ROS describes such a theory. Everybody can learn from scratch the world of robotics and not only that. They can all together contribute as a united community to enhance it with their knowledge and spread their ideas to make it better 2.1.

Robot Operating System is an open source tool to control complex robotic systems. Within its flexible framework it provides services that aim to simplify the task of creating robust, general-purpose robot software. It also provides hardware abstraction, package management, libraries and message-passing processes to accomplish fast, integrated solutions.

ROS packages are containing application-related code which uses libraries such as C++, Python, Lisp and running on Unix-based platforms. Other important libraries to integrate with ROS are Gazebo for simulation and OpenCv for image recognition. The latter will be described thoroughly in the next chapter.

ROS is especially known for its collaborative behavior between developers, researchers and laboratories across the world. Everyone can use and built upon each others work and then distribute to the ecosystem. This effort is supported by ROS packages system and its containing files dependencies but also with Stacks and Repositories 2.2.



Figure 2.2: Plumbing: ROS provides publish-subscribe messaging infrastructure. Tools: ROS provides an extensive set of tools for configuring, debugging, visualizing, logging etc. Capabilities: ROS provides a broad collection of libraries. Ecosystem: ROS is supported and improved by a large community, thousands of ROS packages [1]

2.1.1 History

ROS, Robot Operating System, was first developed in 2007 by Stanford University [1]. ROS had many ancestors and contributors but the first vision of an open source robotics platform was provided by Willow Garage. Later, researchers and developers contributed to extend this concept between the basic

ROS idea and the determinant software packages. Over the years, the contributors placed their code in the same server, making public their repository and that is how the ROS ecosystem was gradually built.

2.1.2 Why Use ROS

There are many software platforms for robotic projects, the most active are listed below.

- MSRDS Microsoft Robotics Developer Studio, Microsoft - U.S.
- ERSP Evolution Robotics Software Platform, Evolution Robotics - Europe
- OpenRTM National Institute of Adv. Industrial Science and Technology (AIST) - Japan
- OROCOS Europe
- OPRoS ETRI, KIST, KITECH, Kangwon National University - South Korea
- NAOqiOS SoftBank and Aldebaran - Japan and France

Each one of them provides unique and convenient functions such as component extension, communication feature, visualization, simulator, real-time and much more. However, much like the current operating systems of personal computers, the robot software platforms that are selected by users will become more popular while others are diminishing. Over the years, ROS development attracts more and more developers and researchers. This section describes why somebody should use ROS specifically [2] [3].

- ROS has a wide variety of packages due to ROS distributed ecosystem. With these packages could be achieved software reuse. Packages include algorithms, applications and hardware, from computing trajectory for mobile robot to control a drone from joystick.
- ROS makes system robust. This means that if one robot's motor crashes, the robot will not stop doing other tasks. This happens due to ROS message system. If one thing crashes due to internal or external disturbance, the system will not crash too.

- ROS has efficient communication. Different components and subsystems might be running at a ROS project and probably they might be written at different language (C++ or Python). This is achieved by ROS message passing.
- ROS is open-source. ROS packages are open source which means that anyone can use them, modify them or add new functionalities without a fee. Also, ROS can integrate with other open source programs, such as gazebo and OpenCv for simulation and image processing respectively.
- ROS supports real time processes and concurrency in robot projects.
- ROS is light. ROS do not need a lot of space in computer's memory and resources.
- ROS is flexible and adaptable to any environment. Robots communication is achieved by message passing between nodes. These nodes run independently and separate from each other which makes robot development more easier. For example, a robot's sensor node sends data as messages to be consumed by any other node. Furthermore, nodes in ROS can be on multiple computers and different architecture. For example, nodes can run either in Raspberry Pi , Arduino or even smartphones.
- ROS has a big community. Researchers and developers from all around the world contribute with new packages or upgrade the existing ones.
- Any large robotics system should support multi-threading processing with multiple robots. This is achieved by ROS message system which simplifies these processes.

2.1.3 Applications Using ROS

ROS was first released in 2007. Since then a lot of different types of robots were developed such as humanoid, mobile, manipulators, drones, underwater vehicles. This gives the opportunity to somebody use this platform to upgrade the existing software without starting from the beginning which could be time consuming. The following examples located in <http://www.ros.org/wiki/Robots> include this variety [1].

Roomba

Roomba is an autonomous vacuum cleaner that was introduced and developed in 2002 by iRobot. Roomba includes sensors such as cliff and dirt sensor to detect dust and avoid obstacles and actuators such as brush, vacuum and wheel motors that helps it navigate through space. Roomba uses node drivers to describe its functionalities such as velocity, odometry and other features from ROS libraries.

Erle-Copter

Erle-copter is a Linux-based quadcopter powered by Erle-brain Linux autopilot. It is a flying kit designed for outdoor operations and it is easily customizable for users to be able to add different components or include modules (camera, gps, anti-vibration system etc). Its flight software communicates with ROS to achieve different flight modes.

NXT-Lego

NXT is a modular robotics kit manufactured by Lego. The main component of the kit is a brick-shaped computer, called the NXT Intelligent Brick, that can take input from sensors and control actuators through a number of microcontrollers. Lego NXT is compatible with ROS providing the ability to run applications such as keyboard/joystick teleoperation, wheel odometry, etc. The bridge between NXT and ROS creates a ROS topic for each motor and sensor of the NXT robot. Users can create their own programs or download the desired software. The NXT-ROS software stack supports several NXT models for users to get familiar with NXT and ROS.

2.1.4 ROS Terminology

In order to develop a project related to ROS, it is necessary to understand the essential components and concepts of ROS. This section will introduce the terminology used in ROS and the important concepts of ROS [5].

ROS

ROS provides standard operating system services such as hardware abstraction, device drivers, implementation of commonly used features including sensing, recognizing, mapping, motion planning, message passing between processes, pack-

age management, visualizers and libraries for development as well as debugging tools.

Master

The master acts as a name server for node-to-node connections and message communication. The command `roscore` is used to run the master. If master has run, it is possible to register the name of each node and get information when needed. The connection between nodes and message communication such as topics and services are impossible without the master.

The master communicates with slaves using XMLRPC (XML-Remote Procedure Call), which is an HTTP-based protocol that does not maintain connectivity. In other words, the slave nodes can access only when they need to register their own information or request information of other nodes. The connection status of each other is not checked regularly. Due to this feature, ROS can be used in very large and complex environments. When ROS is executed, the master will be configured with the URI address (a unique address that represents a resource on the Internet) and port configured in the `ROS_MASTER_URI`. By default, the URI address uses the IP address of local PC, and port number 11311.

Node

A node refers to the smallest unit of processor running in ROS. It is like an executable program. According to ROS, it is better creating one single node for each purpose. For example, in case of mobile robots, the program to operate the robot is broken down into specialized functions. Specialized node is used for each function such as sensor drive, sensor data conversion, obstacle recognition, motor drive, encoder input, and navigation.

Upon startup, a node registers information such as name, message type, URI address and port number of the node. The registered node can act as a publisher, subscriber, service server or service client based on the registered information, and nodes can exchange messages using topics and services.

The node uses XMLRPC for communicating with the master and uses XMLRPC or TCPROS of the TCP/IP protocols when communicating between nodes. Connection request and response between nodes use XMLRPC, and message communication uses TCPROS because it is a direct communication between

nodes independent from the master. As for the URI address and port number, a variable called `ROS_HOSTNAME`, which is stored on the computer where the node is running, is used as the URI address, and the port is set to an arbitrary unique value.

Package

A package is the basic unit of ROS. The ROS application is developed on a package basis, and the package contains either a configuration file to launch other packages or nodes. The package also contains all the files necessary for running the package, including ROS dependency libraries for running various processes, datasets, and configuration file.

Metapackage

A metapackage is a set of packages that have a common purpose. For example, the Navigation metapackage consists of 10 packages including `AMCL`, `DWA`, `EKF`, and `map_server`.

Message

A node sends or receives data between nodes via a message. Messages are variables such as integer, floating point, and boolean. A message can contain other messages. TCPROS and UDPROS communication protocol is used for message delivery. Topic is used in unidirectional message delivery while service is used in bidirectional message delivery that request and response are involved.

Topic

The topic is literally like a topic in a conversation. The publisher node first registers its topic with the master and then starts publishing messages on a topic. Subscriber nodes that want to receive the topic request information of the publisher node corresponding to the name of the topic registered in the master. Based on this information, the subscriber node directly connects to the publisher node to exchange messages as a topic.

Publish and Publisher

The term ‘publish’ stands for the action of transmitting relative messages corresponding to the topic. The publisher node registers its own information and topic with the master, and sends a message to connected subscriber nodes that are interested in the same topic. The publisher is declared in the node and can

be declared multiple times in one node.

Subscribe and Subscriber

The term ‘subscribe’ stands for the action of receiving relative messages corresponding to the topic. The subscriber node registers its own information and topic with the master, and receives publisher information that publishes relative topic from the master. Based on received publisher information, the subscriber node directly requests connection to the publisher node and receives messages from the connected publisher node. A subscriber is declared in the node and can be declared multiple times in one node.

The topic communication is an asynchronous communication which is based on publisher and subscriber, and it is useful to transfer certain data. Since the topic continuously transmits and receives stream of messages once connected, it is often used for sensors that must periodically transmit data. On the other hands, there is a need for synchronous communication with which request and response are used. Therefore, ROS provides a message synchronization method called ‘service’. A service consists of the service server that responds to requests and the service client that requests to respond. Unlike the topic, the service is a one-time message communication. When the request and response of the service is completed, the connection between two nodes is disconnected.

Service

The service is synchronous bidirectional communication between the service client that requests a service regarding a particular task and the service server that is responsible for responding to requests.

Service Server

The ‘service server’ is a server in the service message communication that receives a request as an input and transmits a response as an output. Both request and response are in the form of messages. Upon the service request, the server performs the designated service and delivers the result to the service client as a response. The service server is implemented in the node that receives and executes a given request.

Service Client

The ‘service client’ is a client in the service message communication that requests

service to the server and receives a response as an input. Both request and response are in the form of message. The client sends a request to the service server and receives the response. The service client is implemented in the node which requests specified command and receives results.

Action

The action is another message communication method used for an asynchronous bidirectional communication. Action is used where it takes longer time to respond after receiving a request and intermediate responses are required until the result is returned. The structure of action file is also similar to that of service. However, feedback data section for intermediate response is added along with goal and result data section which are represented as request and response in service respectively. There are action client that sets the goal of the action and action server that performs the action specified by the goal and returns feedback and result to the action client.

Action Server

The ‘action server’ is in charge of receiving goal from the client and responding with feedback and result. Once the server receives goal from the client, it performs predefined process.

Action Client

The ‘action client’ is in charge of transmitting the goal to the server and receives result or feedback data as inputs from the action server. The client delivers the goal to the action server, then receives corresponding result or feedback, and transmits follow up instructions or cancel instruction.

Parameter

The parameter in ROS refers to parameters used in the node. Default values are set in the parameter and can be read or written if necessary. In particular, it is very useful when configured values can be modified in real-time. For example, settings such as USB port number, camera calibration parameters, maximum and minimum values of the motor speed could be specified.

Parameter Server

When parameters are called in the package, they are registered with the parameter server which is loaded in the master.

Catkin

The catkin refers to the build system of ROS. The build system basically uses CMake (Cross Platform Make), and the build environment is described in the ‘CMakeLists.txt’ file in the package folder. The Catkin build system makes it easy to use ROS-related builds, package management, and dependencies among packages.

rosvun

rosvun is the basic execution command of ROS. It is used to run a single node in the package. The node uses the ROS_HOSTNAME environment variable stored in the computer on which the node is running as the URI address, and the port is set to an arbitrary unique value.

rosvaunch

While rosvun is a command to execute a single node, rosvaunch in contrast executes multiple nodes. It is a ROS command specialized in node execution with additional functions such as changing package parameters or node names, configuring namespace of nodes, setting ROS_ROOT and ROS_PACKAGE_PATH, and changing environment variables when executing nodes. rosvaunch uses the ‘*.launch’ file to define which nodes to be executed. The file is based on XML (Extensible Markup Language) and offers a variety of options in the form of XML tags.

CMakeLists.txt

Catkin, which is the build system of ROS, uses CMake by default. The build environment is specified in the ‘CMakeLists.txt’ file in each package folder.

package.xml

An XML file contains package information that describes the package name, author, license, and dependent packages.

2.1.5 Message Communication

In the section above was given an introduction to ROS but not an explanation in detail of how ROS works. In this section, ROS core functions and concepts will be described, as well as detailed description of each terms [5].

As described, ROS is developed in unit of nodes. [10] The node exchanges data with other nodes through messages forming a large program as a whole. The key concept here is the message communication methods among nodes. There are three different methods of exchanging messages: topics, services and actions.

Topics

Communication on topic uses the same type of message for both publisher and subscriber. The subscriber node receives the information of publisher node corresponding to the identical topic name registered in the master. Based on this information, the subscriber node directly connects to the publisher node to receive messages.

As topics are unidirectional and remain connected to continuously send or receive messages, it is suitable for sensor data that requires publishing messages periodically. In addition, one or more subscribers can receive message from one or more publishers.

For example, a robot's camera is to take images for displaying them on laptop screen. For that concept three nodes are necessary, one to contact with camera, one with image processing unit on the robot and one for displaying on laptop screen.

All those nodes are registering to ROS master so they communicate with each other. The camera node subscribes to a specific topic and the other two are publishing on the same as well. This topic serves like storage. So, when the camera node receives data it sends them through ROS messages to the topic above and the other two nodes can get them directly.

Services

Communication on service is a bidirectional synchronous communication between the service client requesting a service and the service server responding to the request. The aforementioned 'publish' and 'subscribe' of the topic is an asynchronous method which is advantageous on periodical data transmission. On the other hand, there is a need for synchronous communication which uses request and response. Accordingly, ROS provides a synchronized message

communication method called ‘service’.

A service consists of a service server that responds only when there is a request and a service client that can send requests as well as receiving responses. Unlike the topic, the service is one-time message communication. Therefore, when the request and response of the service are completed, the connection between two nodes will be disconnected. A service is often used to command a robot to perform a specific action or nodes to perform certain events with a specific condition. Service does not maintain the connection, so it is useful to reduce the load of the network by replacing topic. For example, if the client requests the server for the current time, the server will check the time and respond to the client, and the connection is terminated.

In the robot’s camera concept, the Image Processing Node is not able to request data from the Camera Node. This can be achieved using ROS services. When the Image Processing Node sends a request, the Camera Node gathers data from Camera and then sends the reply.

Actions

Communication on action is used when a requested goal takes a long time to be completed, therefore progress feedback is necessary. This is very similar to the service where ‘goals’ and ‘results’ correspond to ‘requests’ and ‘responses’ respectively. In addition, the ‘feedback’ is added to report feedbacks to the client periodically when intermediate values are needed. The message transmission method is the same as the asynchronous topic. The feedback transmits an asynchronous bidirectional message between the action client which sets the goal of the action and an action server that performs the action and sends the feedback to the action client. Unlike the service, the action is often used to command complex robot tasks such as canceling transmitted goal while the operation is in progress.

A relative example to describe this process could be the control of a tilting laser scanner by an operator. The operator (action client) sends a goal to scanner (action server), to scan an object. This goal includes a signal to begin the process as well as the necessary scan parameters like min-max angle, speed, etc. As long as the operation takes place, the operator gets feedback of the

progress that could be the time left until the scan completes. When the scan is completed, a result is sent to the operator that may contain a point cloud generated by the scanner, the total time that scanner needed to complete the operation, etc.

The publisher, the subscriber, the service server, the service client, the action server, and the action client can be implemented in separate nodes. In order to exchange messages among these nodes, the connection has to be established first with the help of a master. A master acts like a name server as it keeps names of nodes, topics, services and action as well as the URI address, port number and parameters. In other words, nodes register their own information with the master upon launch, and acquire relative information of other nodes from the master. Then, each node directly connects to each other to perform message communication.

2.1.6 ROS communication models differences

Topics, services and actions can all concurrently run on the same robot project but somebody has to choose which one serves a specific task in a better way.

Topics:

- They are receiving continuous data
- They present a many to many connection. One or more publishers/subscribers can connect to a single topic
- They are asynchronous. Data subscribing/publishing may be at any time independent from senders/receivers. Furthermore a particular publisher can not send to a particular subscriber
- They support one way transport. Only publisher decides when data is going to be send.Subscriber can not send a reply

Services:

- They are used for remote procedure calls that terminate quickly. Services block procedure as well as preempting a service may cause problems, so they are not appropriate for long running processes

- They are synchronous
- They present a one to one connection. Services support only one client and one subscriber
- They support bi-directional communication

Actions:

- They are complex procedures for long tasks as they do not block operation and they can be preempted
- They give feedback during task execution

2.1.7 Analyze Message Communication Flow

The most important communication sequence of the master, nodes, topics, services, and action message is presented below [5].

Running the nodes

A master that manages connection information in a message communication between nodes is an essential element that must be run first in order to use ROS 2.3. The ROS master is run by using the ‘roscore’ command and runs the server with XMLRPC. The master registers the name of nodes, topics, services, action, message types, URI addresses and ports for node-to-node connections, and relays the information to other nodes upon request.

roscore

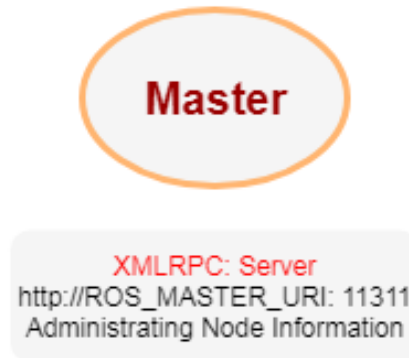


Figure 2.3: Master must be run first in order to use ROS.

Running the Publisher and Subscriber

Publisher and Subscriber nodes are launched with either a 'roslaunch' or 'roslaunch' commands. The nodes register their node name, topic name, message type, URI address, and port with the master as it runs. The master and node communicate using XMLRPC 2.4.

```
$ roslaunch PACKAGE_NAME NODE_NAME
```

```
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```

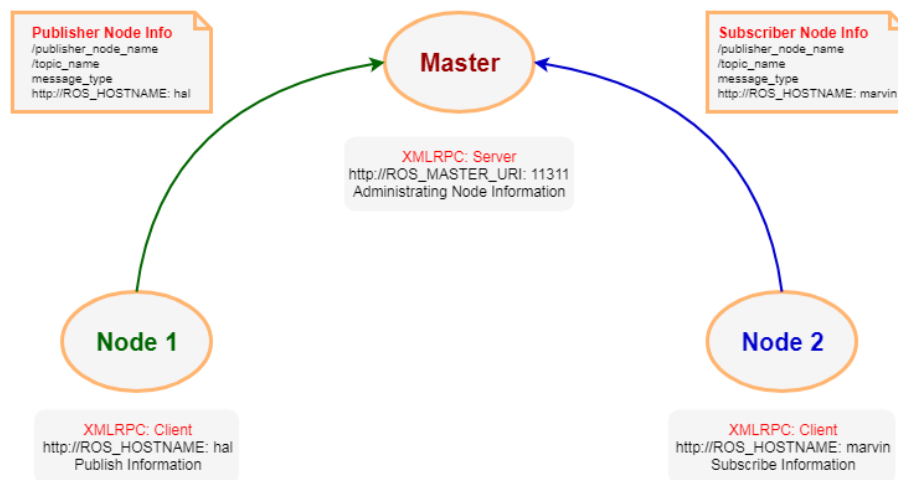


Figure 2.4: Running Publisher/Subscriber

The master distributes the publisher's information to subscribers that want to connect to the publisher node. The master and node communicate using XMLRPC 2.5.

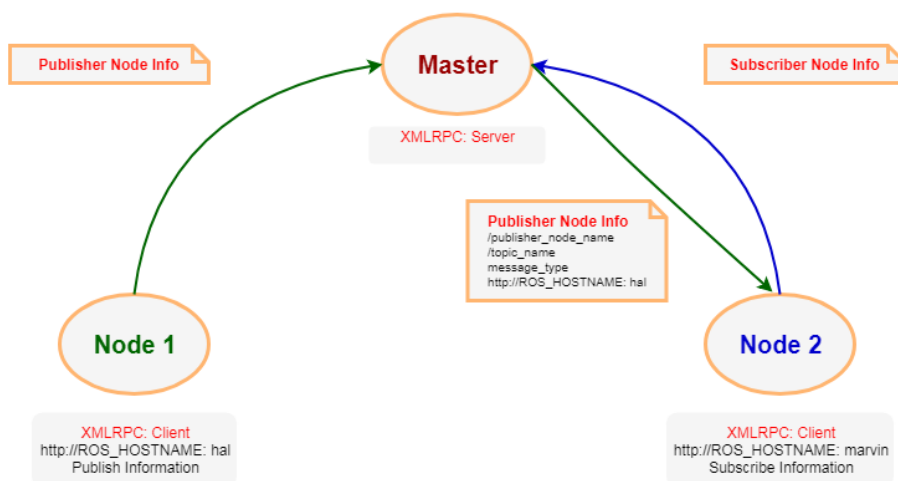


Figure 2.5: Providing publish information

Connection Request/Response from the Subscriber/Publisher Node

The subscriber node requests a direct connection to the publisher node based on the publisher information received from the master. During the request procedure, the subscriber node transmits information to the publisher node such as the subscriber node's name, the topic name, and the message type. The publisher node and the subscriber node communicate using XMLRPC.

The publisher node sends the URI address and port number of its TCP server in response to the connection request from the subscriber node. The publisher node and the subscriber node communicate using XMLRPC.

The subscriber node creates a client for the publisher node using TCPROS, and connects to the publisher node. At this point, the communication between nodes uses TCP/IP based protocol called TCPROS 2.6.

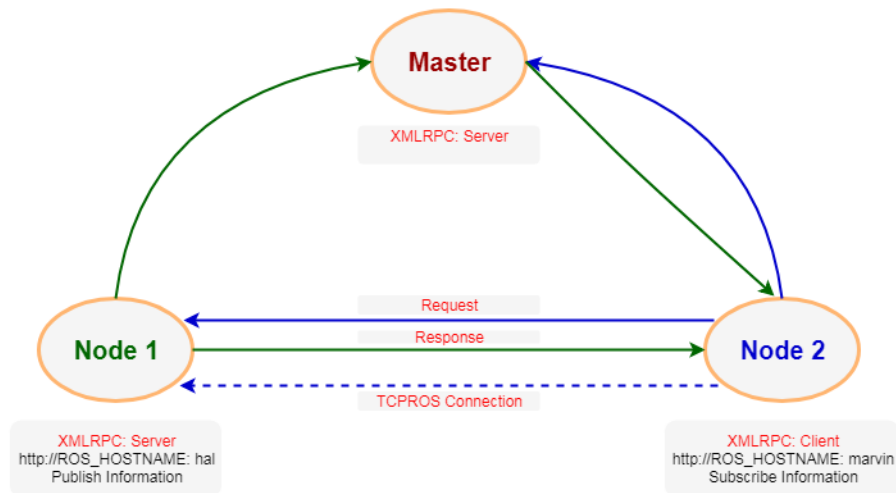


Figure 2.6: Connection Request/Response from the Subscriber/Publisher Node

Message Transmission

The publisher node transmits a predefined message to the subscriber node. The communication between nodes uses TCPROS 2.7.

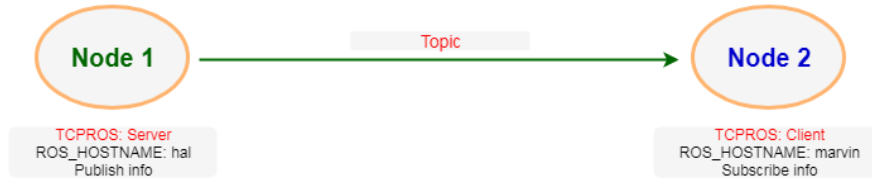


Figure 2.7: Message transmission between nodes

Service Request/Response Connection

The procedures discussed above correspond to the communication on ‘topic’. Topic communication publishes and subscribes messages continuously, unless the publisher or subscriber is terminated. There are two types of services.

-Service Client: Request service and receive response

-Service Server: Receive a service, execute the specified task, and return a response

The connection between the service server and the client is the same as the TCPROS connection for the publisher and subscriber described above. Unlike the topic, the service terminates connection after successful request and response 2.8. If additional request is necessary, the connection procedure must be carried out again.

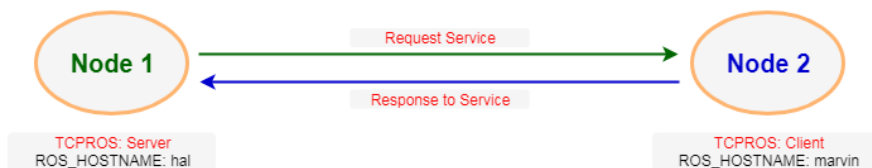


Figure 2.8: Service Connection-Request/Response model

Action Connection

Action may look similar to the request and the response of the service with an additional feedback message in order to provide intermediate result between the request (goal) and the response(result), but in practice it is rather more like a topic 2.9. The connection between the action server and the client is similar to the TCPROS connection of the publisher and subscriber.

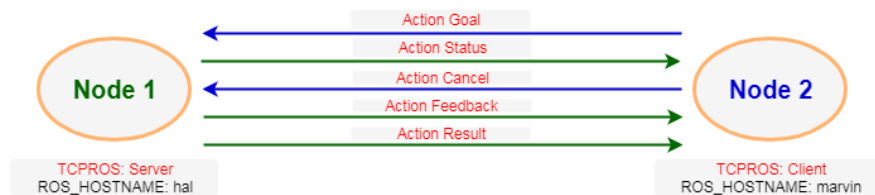


Figure 2.9: Action Connection-Similar to request/response but with additional feedback

2.1.8 Messages

A message is a bundle of data used to exchange data between nodes. The topics, services, and actions are using messages to communicate. A message can include basic data types such as integer, floating point, Boolean as well as message arrays. Moreover, a message can contain other messages. Also, the header 'std_msgs/Header' which is commonly used in ROS can be included in the message. These messages can be described as field types and field names as shown below.

```
fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3
```

```
int32 x
int32 y
```

The topics, services, and actions described in the previous section, use messages.

Although they are similar in the form and the concept, they are divided into three types according to their usage.

msg File

The 'msg' file is the message file used by topics, which has the file extension of '*.msg'. Such msg file consists of field types and field names. A simple example follows the structure of a 'string' message in the 'msg' file which is described below. In this case, a simple type string is used for publishing.

```
msg/String.msg
```

```
string
```

There are also more complicated message structures with nested messages. For example, in the case of the 'teleop_turtle_key' node of the turtlesim package from the turtlesim example, which can be found on www.ros.org.com, the translational speed (meter/sec) and rotational speed (radian/sec) is sent as a message to the turtlesim node according to the directional keys (\leftarrow , \rightarrow , \uparrow , \downarrow) entered from the keyboard. The TurtleBot moves on the screen using the received speed values. The message used at this time is the 'twist' message in 'geometry_msgs'.

```
geometry_msgs/Twist.msg
```

```
Vector3 linear
```

```
Vector3 angular
```

In the message structure below, 'linear' and 'angular' values are declared as a Vector3 type. This is the similar form to the nested message as the Vector3 is a message type in the 'geometry_msgs'. The Vector3 contains the following data.[5]

```
float64 x float64 y float64 z
```

In other words, six topics published from the 'teleop_turtle_key' node are linear.x, linear.y, linear.z, angular.x, angular.y, and angular.z. All of these are float64 type. With these data, arrow keys of the keyboard can be converted

to the translational speed (meter/sec) and the rotational speed (radian/sec) message, so that the TurtleBot could be controlled.

srv File

The 'srv' file is the message file used by services, with the file extension of '*.srv'. For example, the 'AddTwoInts' message in the 'srv', like in the the message structure below, describes a typical srv file. The major difference from the msg file is that the series of three hyphens (—) serve as a delimiter; the upper message being the service request message and the lower message being the service response message. In the example, 'A' and 'B' are the requested integers and Sum is the number that comes from summarize these integers, as response.

```
srv/AddTwoInts.srv
```

```
int64 A
int64 B
---
int64 Sum
```

action File

The action message file is the message file used by actions, with the file extension of '*.action'. Unlike msg and srv, it is relatively uncommon message file, so there is no typical example of the message file, but can be used as shown in following example. The major difference from the msg and srv files is that the series of three hyphens (—) are used in two places as delimiters, the first being the goal message, the second being the result message, and the third being the feedback message. The biggest difference of the action file is the feedback message feature. The goal message and the result message of the action file can be compared to the request and the response message of the srv file mentioned above, but the additional feedback message of the action file is used to send feedback while the designated process is being performed. As describe in the following example, when the starting position of 'start_pose' and the goal position of 'goal_pose' of the robot are transmitted as request values, the robot moves to the received goal

position and returns the 'result_pose'. While the robot is moving to the goal position, the 'percent_complete' message periodically transmits feedback values showing the progress in the form of the percentage of the goal point reached.

```
geometry_msgs/PoseStamped start_pose
geometry_msgs/PoseStamped goal_pose
---
geometry_msgs/PoseStamped result_pose
---
float32 percent_complete
```

2.1.9 ROS Tools

3D Visualization Tool (RViz)

RViz is the 3D visualization tool of ROS 2.10. The main purpose is to show ROS messages in 3D, allowing us to visually verify data. For example, it can visualize the distance from the sensor of a Laser Distance Sensor (LDS) to an obstacle, the Point Cloud Data (PCD) of the 3D distance sensor, the image value obtained from a camera, and many more without having to separately develop the software.



Figure 2.10: Rviz is the 3D visualization tool of ROS [5].

It also supports various visualization using user specified polygons, and Interactive Markers allowing users to perform interactive movements with commands

and data received from the user node. In addition, ROS describes robots in Unified Robot Description Format (URDF), which is expressed as a 3D model for which each model can be moved or operated according to their corresponding degree of freedom, so they can be used for simulation or control. The mobile robot model can be displayed, and received distance data from the Laser Distance Sensor (LDS) can be used for navigation 2.11, 2.12. RViz can also display the image from the camera mounted on the robot. In addition to this, it can receive data from various sensors such as Kinect, LDS, RealSense and visualize them in 3D.

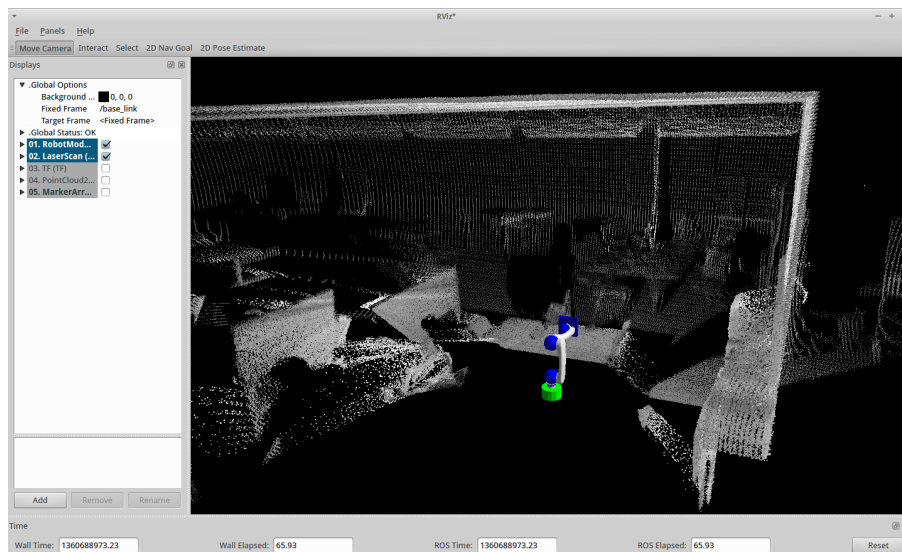


Figure 2.11: View from Laser Distance Sensor[5]

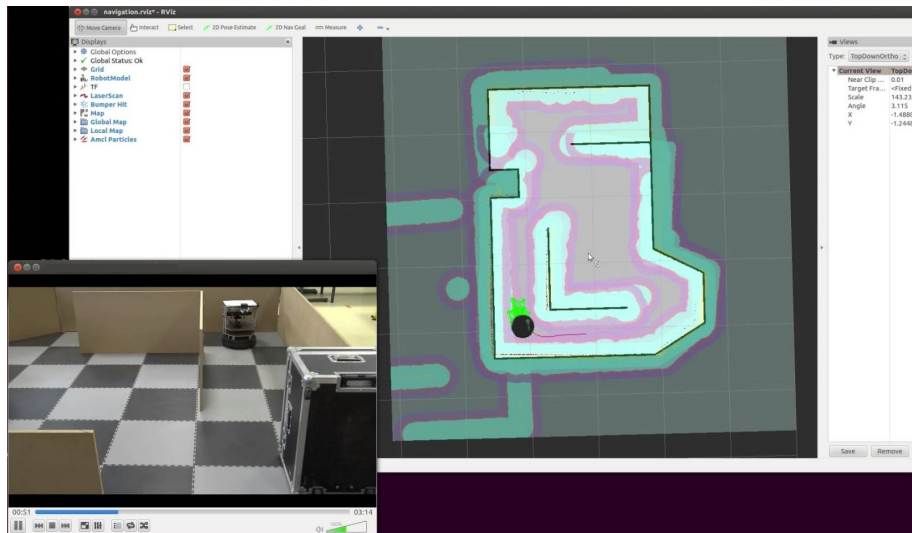


Figure 2.12: Navigation using LDS Sensor[5]

RViz Displays

Display menu is the most frequently used menu to select the message for 3D display 2.13.






























Icon	Name	Description
	Axes	Displays the xyz axes.
	Camera	Creates a new rendering window from the camera perspective and overlays an image on top of it.
	DepthCloud	Displays a point cloud based on the Depth Map. It displays distance values acquired from sensors such as Kinect and Xtion with DepthMap and ColorImage topics as points with overlaid color obtained from the camera.
	Effort	Displays the force applied to each rotary joint of the robot.
	FluidPressure	Displays the pressure of fluids, such as air or water.
	Grid	Displays 2D or 3D grids.
	Grid Cells	Displays each cells of the grid. It is mainly used to display obstacles in the costmap of the navigation
	Group	This is a container for grouping displays. This allows us to manage the displays being used as one group.
	Illuminance	Displays the illuminance.
	Image	Displays the image in a new rendering window. Unlike the Camera display, it does not overlay the camera
	InteractiveMarkers	Displays Interactive Markers. We can change the position (x, y, z) and rotation (roll, pitch, yaw) with the mouse.
	LaserScan	Displays the laser scan value.
	Map	Displays the occupancy map, used in navigation, on top of the ground plane.
	Marker	Displays markers such as arrows, circles, triangles, rectangles, and cylinders provided by RViz.
	MarkerArray	Displays multiple markers.
Icon	Name	Description
	Odometry	Displays the odometry information in relation to the passage of time in the form of arrows. For example, as the robot moves, arrow markers are displayed showing the traveled path in a connected form according to the time intervals.
	Path	Displays the path of the robot used in navigation.
	Point Cloud	Displays point cloud data. This is used to display sensor data from depth cameras such as RealSense, Kinect, Xtion, etc. Since PointCloud2 is compatible with the latest Point Cloud Library (PCL), we can generally use PointCloud2.
	Point Cloud2	Displays point cloud data. This is used to display sensor data from depth cameras such as RealSense, Kinect, Xtion, etc. Since PointCloud2 is compatible with the latest Point Cloud Library (PCL), we can generally use PointCloud2.
	PointStamped	Displays a rounded point.
	Polygon	Displays a polygon outline. It is mainly used to simply display the outline of a robot on the 2D plane.
	Pose	Displays the pose (location + orientation) on 3D. The pose is represented in the shape of an arrow where the origin of the arrow is the position(x, y, z) and the direction of the arrow is the orientation (roll, pitch, yaw). For instance, pose can be represented with the position and orientation of the 3D robot model, while it can be represented with the goal point.
	Pose Array	Displays multiple poses.
	Range	This is used to visualize the measured range of a distance sensor such as an ultrasonic sensor or an infrared sensor in the form of a cone.
	RelativeHumidity	Displays the relative humidity.
	RobotModel	Displays the robot model.
	TF	Displays the coordinate transformation TF used in ROS. It is displayed with the xyz axes much like the previously mentioned axes, but each axis expresses the hierarchy with an arrow according to the relative coordinates.
	Temperature	Displays the temperature.
	WrenchStamped	Displays the wrench, which is the torsion movement, in the form of 'arrow' (force) and 'arrow+circle' (torque).

Figure 2.13: Rviz Displays Panel [5]

ROS GUI Development Tool (rqt)

The rqt tool that is a part of ROS allows graphical representations of ROS nodes, topics, messages, and other information. The ROS wiki lists many of the possible tools that are added to the rqt screen as plugins such as 'rqt' plugins 'rqt_image_view', 'rqt_graph', 'rqt_plot' and 'rqt_bag'.

rqt_image_view

This is a plugin to display the image data of a camera 2.14. Although it is not an image processing process, it is still quite useful for simply checking an image.

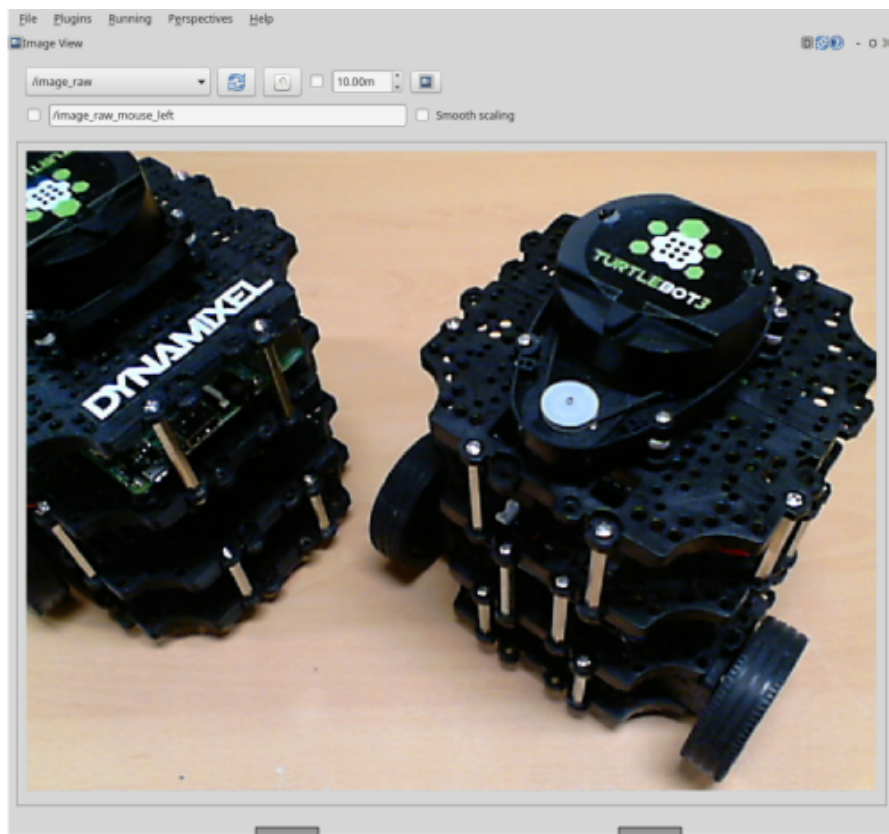


Figure 2.14: Example of rqt_image_view displaying camera data [5]

rqt_graph

The 'rqt_graph' is a tool that shows the correlation among active nodes and messages being transmitted on the ROS network as a diagram. This is very useful for understanding the current structure of the ROS network.

rqt_plot

The 'rqt_plot' is a tool for plotting 2D data 2.15. Plot tool receives ROS messages and displays them on the 2D coordinates.

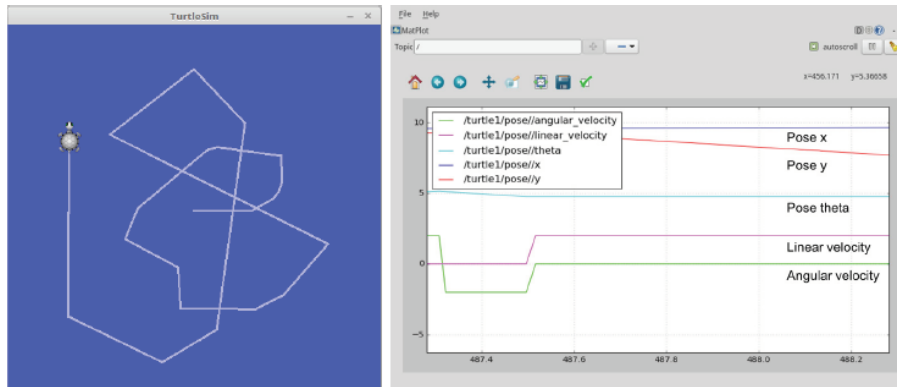


Figure 2.15: Example of rqt_plot converting ROS messages into coordinates [5]

rqt_bag

The 'rqt_bag' is a GUI tool for visualizing a message. It has visualization function added which allows us to see the image of the camera right away, making it very useful for managing image data messages.

rqt_console

The 'rqt_console' is a viewer in the rqt package that displays messages being published to roscout 2.16. It collects messages over time, and lets users view them in more detail, as well as allowing them to filter messages by various means.

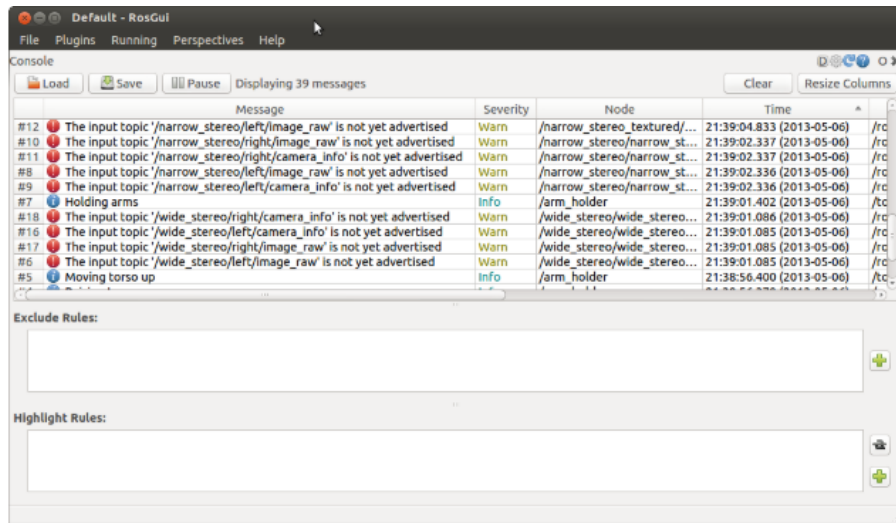


Figure 2.16: Example of rqt_console displaying various messages [5]

ROS public packages

As mentioned above, ROS provides packages which are developed and released by users, ready to be re-used. There are packages for every ROS version and they are located in [1]. In this project were used two of these packages related to camera.

cv_camera

Cv_camera is a ROS camera driver which uses OpenCV capture object to capture images from camera and publish them in topic `/cv_camera/image_raw`.

```
$ rosrun cv_camera cv_camera_node
```

image_saver

Image_saver is a tool from plugin image_view that allows users to save images as jpg/png file from streaming topic. Image_saver subscribes to the streaming topic e.g. `/cv_camera/image_raw` and when rosservice call `/image_saver/save` is called image_saver grabs a picture from camera.

```
$ rosrun image_view image_saver image:=/cv_camera/image_raw
```

2.2 SSH Definition

Secure Socket Shell or SSH is a network protocol that provides administrators with a secure way to access a remote computer [17]. Ssh connection starts a client program that enables secure connection to a server on a remote machine. This method provides an encrypted connection between two hosts over an insecure network. After setting up this connection it is possible to execute commands or transfer files on the remote server [14].

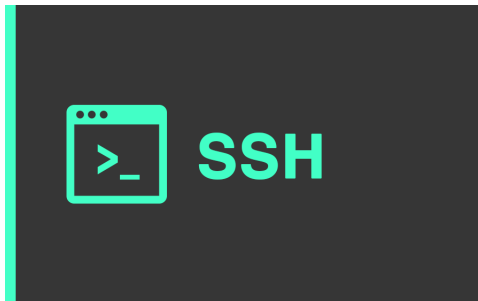


Figure 2.17: SSH is a network protocol that provides administrators with a secure way to access a remote computer [18].

2.2.1 Set up ssh keys

In order to be able to start a secure SSH connection between two machines, the following configuration is required [16].

- Create a key pair, private and public

```
$ ssh-keygen -t rsa
```

- Store them in a local machine's file location

- Enter a passphrase (optionally). The user decides if he wants more security in his connection if private key falls in unauthorized possession
- Copy public key in new machine's `authorized_keys` file with the following command.

```
$ ssh-copy-id
```

2.2.2 How it works

The SSH protocol employs a client-server model to authenticate two parties and encrypt the data between them [17].

The server component listens on a designated port for connections. It is responsible for negotiating the secure connection, authenticating the connecting party, and spawning the correct environment if the credentials are accepted.

The client is responsible for beginning the initial TCP handshake with the server, negotiating the secure connection, verifying that the server's identity matches previously recorded information, and providing credentials to authenticate.

An SSH session is established in two separate stages. The first is to agree upon and establish encryption to protect future communication. The second stage is to authenticate the user and discover whether access to the server should be granted 2.18.

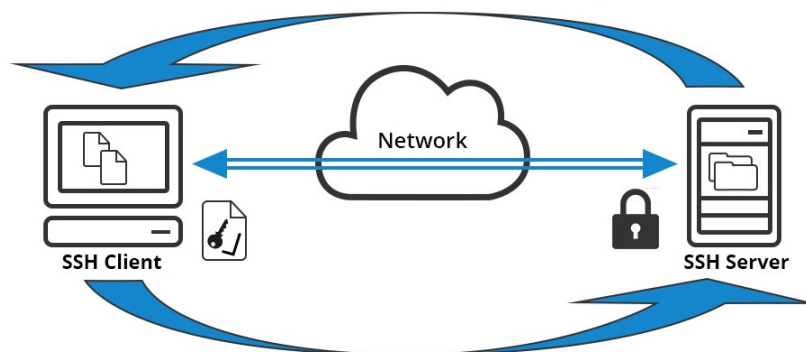


Figure 2.18: The SSH protocol employs a client-server model to authenticate two parties and encrypt the data between them [19].

SCP - Secure File Copy

scp is a program for copying files between computers. It uses the SSH protocol. It is included by default in most Linux and Unix distributions [14].

The basic usage of scp is as follows:

```
$ scp file host:path
```

```
$ scp host:file path
```

The first line copies the file to the remote host in a specific path. Alternatively, the second line fetches the file from the remote host, and puts it in the directory indicated by path.

2.3 Database

Database, also called electronic database, any collection of data, or information, that is specially organized for rapid search and retrieval by a computer. Databases are structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations. A database management system (DBMS) extracts information from the database in response to queries [20].

A database is stored as a file or a set of files on magnetic disk or tape, optical disk, or some other secondary storage device. The information in these files may be broken down into records, each of which consists of one or more fields. Fields are the basic units of data storage, and each field typically contains information pertaining to one aspect or attribute of the entity described by the database. Records are also organized into tables that include information about relationships between its various fields. Although database is applied loosely to any collection of information in computer files, a database in the strict sense provides cross-referencing capabilities. Using keywords and various sorting commands, users can rapidly search, rearrange, group, and select the fields in many records to retrieve or create reports on particular aggregates of data.

Database records and files must be organized to allow retrieval of the information. Queries are the main way users retrieve database information. The power of a DBMS comes from its ability to define new relationships from the basic ones given by the tables and to use them to get responses to queries. Typically, the user provides a string of characters, and the computer searches the database for a corresponding sequence and provides the source materials in which those characters appear; a user can request, for example, all records in which the contents of the field for a person's last name is the word Snow.

The many users of a large database must be able to manipulate the information within it quickly at any given time. Moreover, large business and other organizations tend to build up many independent files containing related and even overlapping data, and their data-processing activities often require the linking of data from several files. Several different types of DBMS have been developed to support these requirements: flat, hierarchical, network, relational, and object-oriented.

2.3.1 Evolution of Databases

Databases have evolved since their inception in the 1960s, beginning with hierarchical and network databases, through the 1980s with object-oriented databases, and today with SQL and NoSQL databases and cloud databases.

In one view, databases can be classified according to content type: bibliographic, full text, numeric and images. In computing, databases are sometimes classified according to their organizational approach. There are many different kinds of databases 2.19, ranging from the most prevalent approach, the relational database, to a distributed database, cloud database or NoSQL database [22].

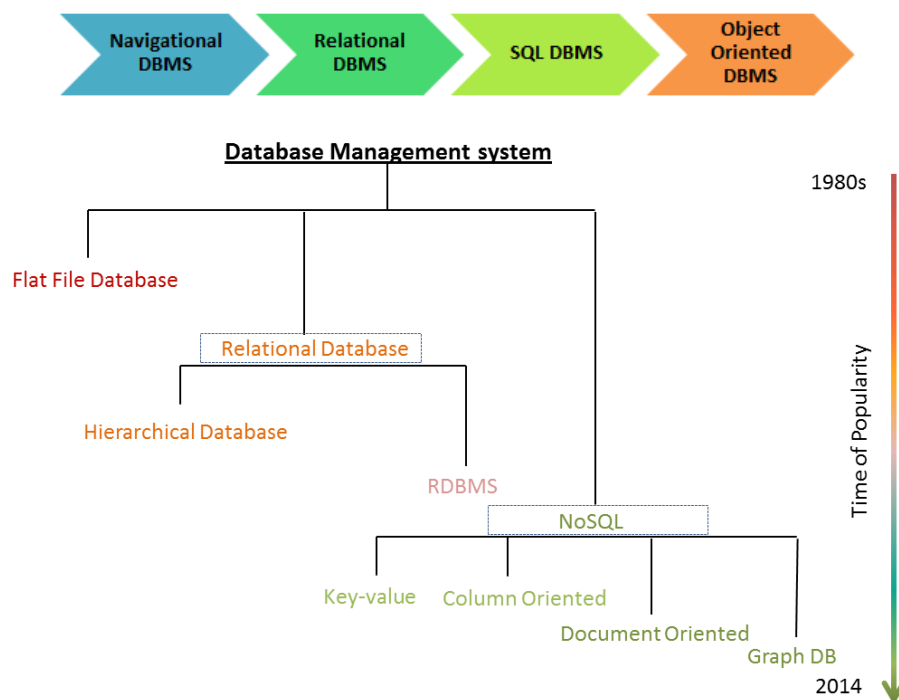


Figure 2.19: Database's evolution [21] [23]

Types of DBMS

There are four major types of DBMS [21].

- *Hierarchical* - this type of DBMS employs the "parent-child" relationship of storing data. This type of DBMS is rarely used nowadays. Its structure is like a tree with nodes representing records and branches representing fields. The windows registry used in Windows XP is an example of a hierarchical database. Configuration settings are stored as tree structures with nodes.
- *Network DBMS* - this type of DBMS supports many-to-many relations. This usually results in complex database structures. RDM Server is an example of a database management system that implements the network model.

- *Relational DBMS* - this type of DBMS defines database relationships in form of tables, also known as relations. Unlike network DBMS, RDBMS does not support many to many relationships. Relational DBMS usually have pre-defined data types that they can support. This is the most popular DBMS type in the market. Examples of relational database management systems include MySQL, Oracle, Sqlite and Microsoft SQL Server database.
- *Object Oriented Relation DBMS* - this type supports storage of new data types. The data to be stored is in form of objects. The objects to be stored in the database have attributes (i.e. gender, age) and methods that define what to do with the data. PostgreSQL is an example of an object oriented relational DBMS.

2.3.2 SQL Definition

Structured Query Language (SQL) 2.20 is a computer language for database management and data manipulation. SQL is used to query, insert, update and modify data [24].



Figure 2.20: SQL is a computer language for database management and data manipulation [29].

The SQL language is based on several elements. For the convenience of SQL developers, all necessary language commands in the corresponding database

management systems are usually executed through a specific SQL command-line interface (CLI) [25].

- Clauses - the clauses are components of the statements and the queries
- Expressions - the expressions can produce scalar values or tables, which consist of columns and rows of data
- Predicates - they specify conditions, which are used to limit the effects of the statements and the queries, or to change the program flow
- Queries - a query will retrieve data, based on a given criteria
- Statements - with the statements one can control transactions, program flow, connections, sessions, or diagnostics. In database systems the SQL statements are used for sending queries from a client program to a server where the databases are stored. In response, the server processes the SQL statements and returns replies to the client program. This allows users to execute a wide range of amazingly fast data manipulation operations from simple data inputs to complicated queries.

SQL Elements

SQL is a language designed to store data, but the data stored in an SQL database is not static. It can be modified at any time with the use of several very simple commands. The SQL syntax is pretty much self explanatory, which makes it much easier to read and understand.

Queries

The SQL queries are the most common and essential SQL operations. Via an SQL query, one can search the database for the information needed. SQL queries are executed with the “SELECT” statement. An SQL query can be more specific, with the help of several clauses:

- FROM - it indicates the table where the search will be made
- WHERE - it's used to define the rows, in which the search will be carried. All rows, for which the WHERE clause is not true, will be excluded

- ORDER BY - this is the only way to sort the results in SQL. Otherwise, they will be returned in a random order

Example:

```
SELECT * FROM Table1 WHERE number=1 ORDER BY LastName
```

Data Manipulation

Data manipulation is essential for SQL tables - it allows users to modify an already created table with new information, update the already existing values or delete them.

- With the INSERT statement, new rows to an already existing table could be added. New rows can contain information from the start, or can be with a NULL value.

```
INSERT INTO table(name1, name2, name3)
↳ VALUES('value1', 'value2', 'value3');
```

- With the UPDATE statement, the already existing information in an SQL table could be easily be modified.

```
UPDATE table SET number1 = 'true' WHERE firstcolumn = 'number2';
```

- With the DELETE statement, unneeded rows from a table could be removed.

```
DELETE FROM table WHERE firstcolumn = 'value1' AND
↳ secondcolumn = 'value2';
```

Data Definition

Data definition allows user to define new tables and elements.

- CREATE - with the CREATE statement, a new table in an existing database could be created.

```
CREATE TABLE Persons(phone VARCHAR(32), firstname
↳ VARCHAR(32), lastname VARCHAR(32), address VARCHAR(64));
```

Data Control

SQL allows the user to define the access each of the table users can have to the actual table.

- GRANT - with the GRANT statement, users are authorized to modify the selected table.

```
GRANT ALL PRIVILEGES ON database_name TO database_user;
```

2.4 OpenCv

Computer vision and image processing are two fields of technology that are growing rapidly nowadays. This is partly a result of both cheaper and more capable cameras, partly because of affordable processing power, and partly because vision algorithms are starting to mature [35].

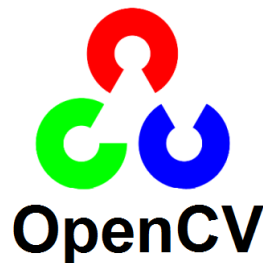


Figure 2.21: OpenCv is an open source real time computer vision library [32].

There is a wide variety of image processing programs such as OpenCv and TensorFlow. OpenCV itself has played a role in the growth of computer vision by enabling thousands of people to do more productive work in vision 2.21. With its focus on real-time vision, OpenCV helps students and professionals efficiently implement projects and jump-start research by providing them with a computer vision and machine learning infrastructure that was previously available only in a few mature research labs.

2.4.1 OpenCv Definition

OpenCv is an open source real time computer vision library. It was first released in 2000 and it is written in C++ but with different bindings with Python, Matlab and Java. It runs on Linux, Windows and Mac OS. OpenCV was designed for computational efficiency and with a strong focus on realtime applications. OpenCV is written in optimized C and can take advantage of multicore processors [35].

One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-inhand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL). This sub-library is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV's mission, but it is general enough to be used for any machine learning problem.

Since its alpha release in January 1999, OpenCV has been used in many applications, products, and research efforts. These applications include stitching images together in satellite and web maps, image scan alignment, medical image noise reduction, object analysis, security and intrusion detection systems, automatic monitoring and safety systems, manufacturing inspection systems, camera calibration, military applications, and unmanned aerial, ground, and underwater vehicles. It has even been used in sound and music recognition, where vision recognition techniques are applied to sound spectrogram images [35].

2.4.2 Computer Vision

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do [31].

The goal of Computer Vision is to emulate human vision using digital images through three main processing components, executed one after the other [30]:

- Image acquisition
- Image processing
- Image analysis and understanding

As our human visual understanding of world is reflected in our ability to make decisions through what we see, providing such a visual understanding to computers would allow them the same power.

Image acquisition

Image acquisition is the process of translating the analog world around us into binary data composed of zeros and ones, interpreted as digital images.

Different tools have been created to build such datasets:

- Webcams and embedded cameras
- Digital compact cameras and DSLR
- Consumer 3D cameras and laser range finders

Most of the time, the raw data acquired by these devices needs to be post-processed in order to be more efficiently exploited in the next steps.

Image processing

The second component of Computer Vision is the low-level processing of images. Algorithms are applied to the binary data acquired in the first step to infer low-level information on parts of the image. This type of information is characterized by image edges, point features or segments, for example. They are all the basic geometric elements that build objects in images.

This second step usually involves advanced applied mathematics algorithms and techniques.

Low-level image processing algorithms include:

- Edge detection

- Segmentation
- Classification
- Feature detection and matching

Image analysis and understanding

The last step of the Computer Vision pipeline is the actual analysis of the data, which will allow the decision making. High-level algorithms are applied, using both the image data and the low-level information computed in previous steps.

Examples of high-level image analysis are:

- 3D scene mapping
- Object recognition
- Object tracking

Applications of computer vision

Techniques developed for Computer Vision have many applications in the fields of robotics, human-computer interaction and visualization, to name a few:

- Motion recognition
- Augmented reality
- Autonomous cars
- Domestic/service robots
- Image restoration such as denoising

When developing Computer Vision algorithms, one has to face different issues and challenges, related to the very nature of the data or even the application to be created and its context:

- Noisy or incomplete data
- Real-time processing

- Limited resources: power, memory

Current research is focused on addressing these challenges to make the algorithms more robust and efficient in difficult conditions.

2.4.3 Haar-Cascade Detection in OpenCv

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images [36].

Cascading is a particular case of ensemble learning based on the concatenation of several Classifiers, using all information collected from the output from a given classifier as additional information for the next classifier in the cascade. Cascading Classifiers are trained with several hundred "positive" sample views of a particular object and arbitrary "negative" images of the same size. After the classifier is trained it can be applied to a region of an image and detect the object in question. To search for the object in the entire frame, the search window can be moved across the image and check every location for the classifier [41].

This process is most commonly used in image processing for object detection and tracking, primarily facial detection and recognition. The first cascading classifier is the face detector of Viola and Jones (2001). [37] The requirement for this classifier was to be fast in order to be implemented on low-power CPUs, such as cameras and phones [41].

In order to train a classifier it needs samples, which means it is needed a lot of images that show the desired object for detection (positive sample) and even more images without the object (negative sample). The best results came from positive images that look exactly like the ones where the desired object is in, except that they are cropped so only the object is visible [39].

A highly accurate classifier should be trained with a lot of negative images that look exactly like the positive ones with the difference that they do not contain the desired object. For example if somebody wants to detect a "stop" sign, the negative images would ideally be a lot of pictures of other signs [39].

The process starts by extracting Haar features from each image as shown below:

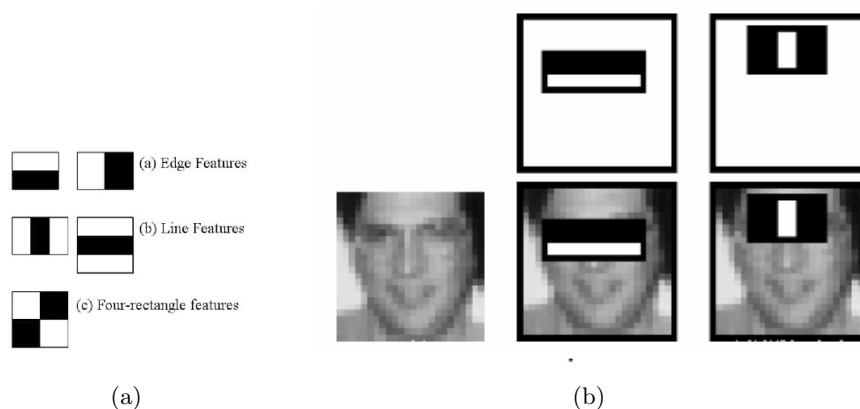


Figure 2.22: Feature Extraction [32]

Each feature is a single value obtained by subtracting sum of pixels under white rectangle from the sum of pixels under black rectangle. Rectangle features can be computed very rapidly using an intermediate representation for the image which called the integral image. It simplifies calculation of sum of pixels, no matter how large may be the number of pixels, to an operation involving just four pixels [36].

For example, in figure 2.22, two features are being extracted. The first one focuses on the property that the region of the eyes is often darker than the area of the nose and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose [40].

But among all these features calculated, most of them are irrelevant. For example, when used on the cheek, the windows become irrelevant because none of these areas are darker or lighter than other regions on the cheeks, all sectors here are the same. So we promptly discard irrelevant features and keep only those relevant with a fancy technique called Adaboost. AdaBoost is a training process which selects only those features known to improve the classification (face/non-face) accuracy of our classifier [40].

The first step is to apply each and every feature on all the training images. For each feature, it will find the best threshold to classify if the faces present or not. There will be errors or misclassifications. By selecting the features with minimum error rate, which means they are the features that best classifies faces.

Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. New error rates are calculated and so as new error rates. This loop will continue until required accuracy or error rate is achieved or required number of features are found. Final classifier is created by combining all of the weak classifiers (they are called weak classifiers because them alone can not classify the image but together they form a strong classifier) [36].

In an image, most of the image region does not contain any faces. It is a better idea to have a simple method to check if a window is not a face region, discard and do not process it. This can speed up the process of face recognition by focusing on the face region only instead of all areas. The method here is using the concept of Cascade of Classifiers. Instead of applying all training features, group the features into different stages of classifiers and apply one-by-one. If a window fails the first stage, it will be discarded immediately, if it passes, apply the second stage of features, keep continuing the loop. Finally, the window which passes all stages is a face region. This explains how Face Detection using Haar Cascade works [36].

2.4.4 YOLO

YOLO (You Only Look Once) is a network for object detection. The object detection task consists in determining the location on the image where certain objects are present, as well as classifying those objects. Previous methods for this, like in section above, execute this task in multiple steps. This can be slow to run and also hard to optimize, because each individual component must be trained separately. YOLO does it all with a single neural network [34].

This neural network splits the image in multiple grid cells with bounding boxes as shown in figure 2.23.

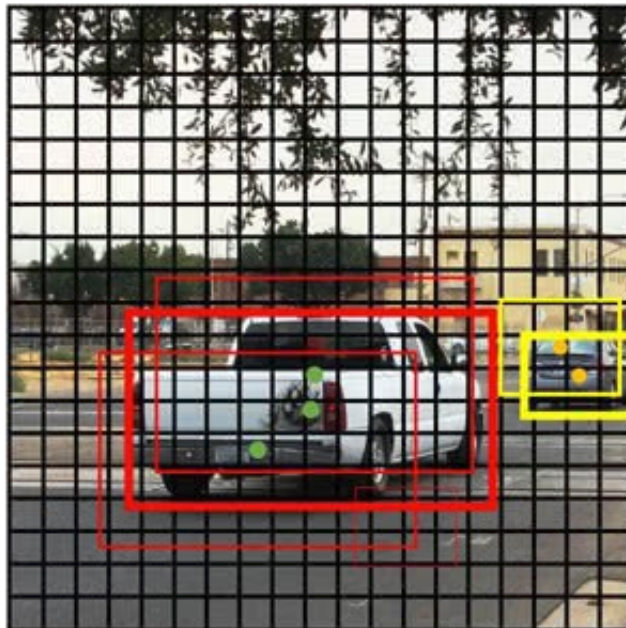


Figure 2.23: Grid Cells with Bounding Boxes

If the grid cell contains an object then the algorithm generates two bounding boxes for each grid cell. The YOLO algorithm takes the middle point of the bounding box and associates it to the grid cell containing it. After all bounding boxes have generated, yolo uses a class probability theory in order to eliminate the boxes that intersect and keep only the prevailing ones those with the higher probability with object inside. This can be done with the non_max suppression method. Non-max suppression is a common algorithm used for cleaning up when multiple boxes are predicted for the same object [42].

For the example below, the model outputs three predictions for the truck in the center. There are three bounding boxes, but we only need one. The thicker the predicted bounding box, the more confident the prediction is that means a higher pc value. The goal is to remove those “shadow” boxes surrounding the main predicted box [42].

Finally, the cleaned up prediction is showed in figure 2.24:

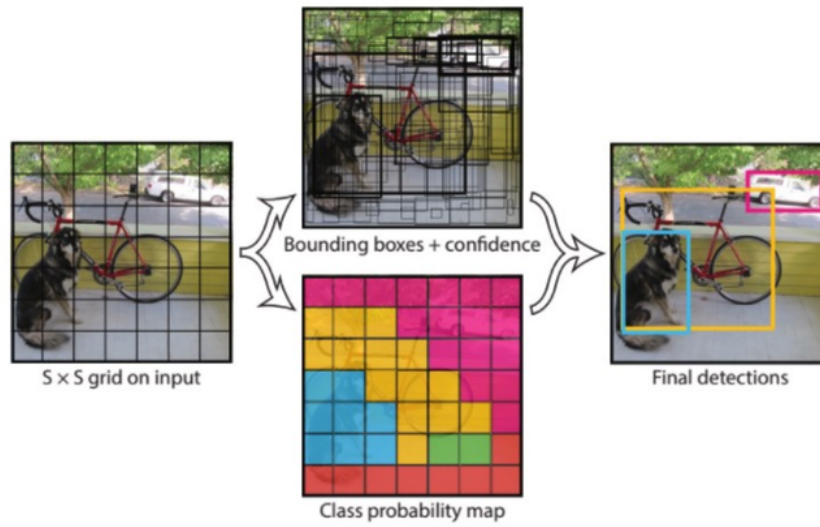


Figure 2.24: Prediction Process [38]

So here is the graph illustrating the prediction process figure 2.25:

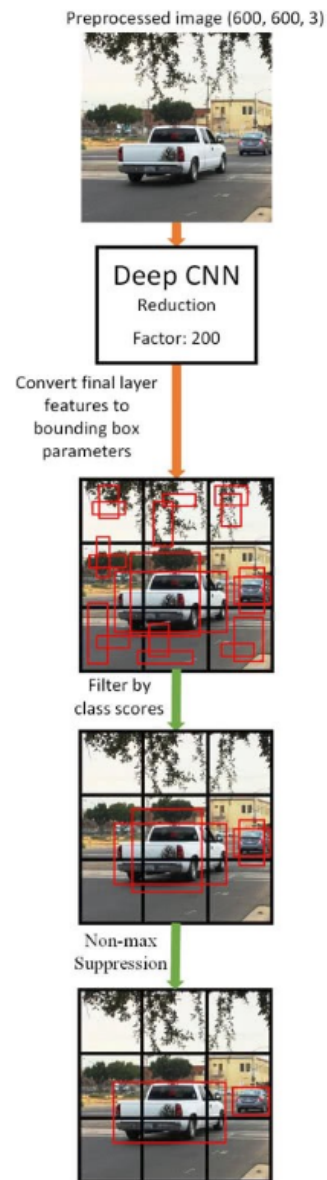


Figure 2.25: Prediction Process [42]

2.4.5 OpenCv in ROS

ROS provides integration with OpenCV to manipulate the images sending between nodes. So, ROS offers a bridge class to transform ROS images back and forth from OpenCV 2.26. The `cv_bridge` library converts a `cv` image into a ROS image message and then republishes it over ROS [1].

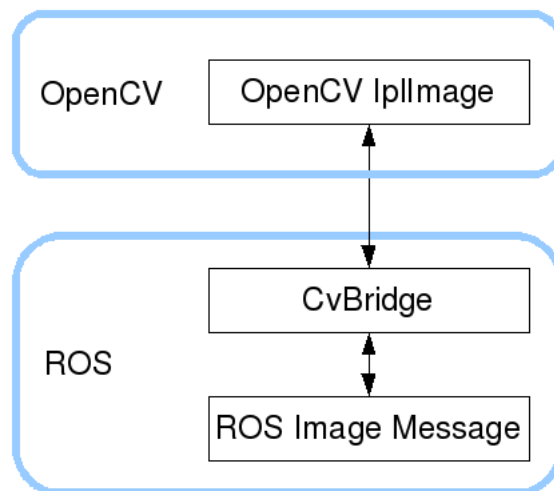


Figure 2.26: Cv_bridge concept of converting type of messages [1]

2.5 Hypertext Markup Language - HTML

2.5.1 HTML Definition

HTML or Hypertext Markup Language is a computer language which is used to create web pages and web applications 2.27.



Figure 2.27: HTML is the standard markup language for Web pages [47].

HTML documents consist of document tags which act to directly describe the visual appearance of a web page or to provide a directive command such as inserting imagery or a link to another web page within a document. HTML documents are saved in text format and are designed to be viewed or edited on any operating system that is able to connect to the Internet [46].

The definition can split in three parts as follows [45].

- Hypertext is a method that lets somebody move around internet by clicking the page he want to enter
- Markup is what the HTML tags define about the text inside them
- Language - HTML is a language because it needs coding and syntax to use

2.5.2 How HTML works

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. Each page contains a series of connections to other pages called hyperlinks.

HTML code ensures the proper formatting of text and images so that your Internet browser may display them as they are intended to look. Without HTML, a browser would not know how to display text as elements or load images or other elements. HTML also provides a basic structure of the page, upon which Cascading Style Sheets are overlaid to change its appearance. One

could think of HTML as the bones (structure) of a web page, and CSS as its skin (appearance) [44].

HTML Markup

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML documents are defined by tags, written using angle brackets, that consist of an opening and closing tag in order to create structured documents. Tags such as `` and `<input />` directly introduce content into the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.[43] A typical HTML document starts with an `<html>` tag and ends with an `</html>` tag. This tag declares the document to be type HTML to a web browser. Most HTML tags are paired, such as `<html> </html>` and `<body> </body>`. An HTML document usually consists of a head, embedded between `<head>` and `</head>` tags, followed by a body, embedded between `<body>` and `</body>` tags. The head usually states the title of the document, which is specified between a pair of `<title>` and `</title>` tags. A `<TITLE>` tag can be displayed in the top of a web browser and can also contain Javascript and other meta data information for the web page. The body is the content of the document. Detailed information, such as text and tables, appears in the body.

The major HTML elements that provide conceptual structure in HTML documents are titles, sections, paragraphs, links, images, lists, tables, forms.

Titles

The title located in the head of the HTML document is converted to an attributed object in the HTML-CM conceptual model. For example, consider the following HTML code 2.28.

```
<html>
<head>
<title>Robot Operating System</title>
</head>

<body>
The content of the document
</body>

</html>
```

Figure 2.28: Html title tags

This code is converted to the following attributed object.

Title: Robot Operating System

Sections

A section in an HTML document is converted to an attributed object. The heading of the section is converted to an attribute and the rest of the section is converted to a value for this attribute.

Paragraphs

A paragraph is the content after a <p> tag up to a </p> tag or another <p> tag. As mentioned in Chapter 1, the </p> tag is optional. If a paragraph has some boldface or italic words or some words preceding a colon at the beginning, then the paragraph is converted to an attributed object, and otherwise it is converted to a list object. A sequence of multiple paragraphs is converted to a list object too. The following is an example for paragraph as well as section 2.29.

```
<html>
<body>
<section>
  <h1>Robot Operating System</h1>
  <p>The Robot Operating System (ROS) is a set of software
libraries and tools that help you build robot applications. </p>
</section>
</body>
</html>
```

(a) Html paragraph tags

Robot Operating System

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications.

(b) Result

Figure 2.29: Example of Html Sections and Paragraphs

Links

A link in an HTML document is converted to a linking object, which consists of a label and an anchor. The label is the text presented and the anchor is the destination of the link, which can refer to another HTML document, a PDF file, a video/audio clip, etc. The following is an example 2.30.


```
<html>
<body>

<h2>HTML Links</h2>
<p><a href="https://www.ros.org/html/">Robot
Operating System</a></p>
</body>
</html>
```

(a) Html links tags

HTML Links

[Robot Operating System](https://www.ros.org/html/)

(b) Result

Figure 2.30: Example of Html Links

Images

A reference to an image document in an HTML document is converted to an attributed object using the Image keyword as the attribute and the destination of the link to the image as the value. The following is an example 2.31.

```
<html>
<body>

<h2>HTML Image</h2>


</body>
</html>
```

(a) Html image tags

HTML Image



(b) Result

Figure 2.31: Example of Html image

Lists

The HTML element is used to represent an item in a list. It must be

contained in a parent element: an ordered list (), an unordered list (), or a menu (<menu>). In menus and unordered lists, list items are usually displayed using bullet points. In ordered lists, they are usually displayed with an ascending counter on the left, such as a number or letter 2.32 2.33.

<code></code>	
<code>Coffee</code>	1. Coffee
<code>Tea</code>	2. Tea
<code>Milk</code>	3. Milk
<code></code>	
(a) Html list tags	(b) Result

Figure 2.32: Example of Html ordered list

<code></code>	
<code>Coffee</code>	• Coffee
<code>Tea</code>	• Tea
<code>Milk</code>	• Milk
<code></code>	
(a) Html list tags	(b) Result

Figure 2.33: Example of Html unordered list

HTML also supports description lists. A description list is a list of terms, with a description of each term. The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term 2.34.

<code><dl></code>	
<code><dt>Coffee</dt></code>	Coffee
<code><dd>- black hot drink</dd></code>	- black hot drink
<code><dt>Milk</dt></code>	Milk
<code><dd>- white cold drink</dd></code>	- white cold drink
<code></dl></code>	
(a) Html list tags	(b) Result

Figure 2.34: Example of Html Description List

Tables

Tables in HTML are used either to display tabular information or to provide visitors with a better visual layout. A table begins with a `<table>` tag and ends with a `</table>` tag. A table that has a caption embedded between `<caption>` and `</caption>` tags is converted to an attributed object by using the caption as the attribute and the rest of the table's contents as the value. The content of the table, ignoring the caption, consists of rows and columns of cells. The `<tr>` tag shows the beginning of a new row. The `<td>` tag shows the beginning of a cell. The `</tr>` and `</td>` tags can be used to end rows and cells, respectively, but they are not mandatory. Both rows and columns can have headings that are specified by `<th>` tags 2.35.

```

<table>
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table>

```

(a) Html table tags

Month	Savings
January	\$100
February	\$50

(b) Result

Figure 2.35: Example of Html Tables

Forms

An HTML form is used to display data and receive input from the user. A form begins with `<form>` and ends with `</form>`. An HTML form is converted to a single attributed object. HTML forms have many special elements called controls. Some example controls are text input, checkbox, button, radio button, and menu. Although the visualizations of these controls are quite different, they all have three common attributes, specifically, names, values, and types. The name specifies the data being displayed as output, the value is the data that are stored in the HTML document, and the type determines the type of value

being received as input. The two methods of sending information from a form to the server are GET and POST. The major difference between them is the way that they organize and send information to the server. The GET method sends the information as name-value pairs. It has a limitation on how long the string can be, which makes it good for sending short information. The POST method first places the name-value pairs together in a file and then sends the file to the server. There is no limit on the size of the file.

The two special types of buttons in HTML forms are SUBMIT and RESET. Activating a SUBMIT button sends information entered so far on the form, while activating a RESET button removes any information entered by the user from the form and prepares the form for new entries. The HTML `<input>` element is used to create interactive controls for web-based forms in order to accept data from the user 2.36.

```
<form action="/example.html">  
  First name:<br>  
  <input type="text" name="firstname" value="John">  
  <br>  
  Last name:<br>  
  <input type="text" name="lastname" value="Snow">  
  <br><br>  
  <input type="submit" value="Submit">  
</form>
```

(a) Html form tags

First name:

Last name:

(b) Result

Figure 2.36: Example of Html Forms

Chapter 3

Implementation

So far the tools including ROS, OpenCV, HTML, Sql, used for the project implementation have been analyzed at a theoretical level. All of them play an equally important role in the composition of the project. This chapter is divided in two sections, the first part contains a description of the basic code structure of each tool separately, which clarifies their functionality. In the second part, it becomes concrete how these tools are linked to each other in order to form an integrated application.

3.1 ROS

3.1.1 ROS Communication models

Topics, services and actions are ROS communication models that have been described and compared in detail in 2.1.5. Before the development of this application is explained, it is important to mention which of these three models was used. The application's main feature requires a one to one, remote and periodic connection in which two locations need to be synchronized and interact with each other. In order to meet the application's requirements, service is selected as the prevailing communication model.

An autonomous device (Unit point) sends periodic messages to a remote station (Base point) where the messages are evaluated by a human operator. This kind

of communication is accomplished by a simple service method (UnitCall). Additionally, the operator has the ability to request information about the device's status at any time, which requires the use of an additional service (BaseCall). Each service consists of a client node, a server node and a service file.

The application also required a continuous communication with the Unit's camera data stream. Data might be published and subscribed at any time independent of any senders/receivers. This was achieved by using topics communication model.

These two client/server models as well as the topic communication model will be further analyzed in the sections below.

Package

First of all, after installing ROS, a directory named `catkin_ws` (catkin workspace) will be created in order to build ROS packages. The command `catkin_init_workspace` will setup this directory to be able to build ROS packages.

Afterwards, the following command creates the packages used for the project implementation inside the workspace's `src` folder.

```
$ cd ~/catkin_ws/src
```

```
$ catkin_create_pkg application message_generation std_msgs rospy  
→ cv_bridge
```

The 'application' package was created with dependencies: 'message_generation', that creates a new message, 'std_msgs', that is the ROS standard message package, 'rospy' that allows the client library to use Python in ROS and 'cv_bridge' that allows using cv_bridge package. These packages may also be added after creating the 'package.xml' file.

In the 'application' folder, the 'CMakeLists.txt' and 'package.xml' files are created along with default folders.

```
$ cd application
```

```
$ ls
```

<code>include</code>	Header File Folder
<code>src</code>	Source Code Folder
<code>CMakeLists.txt</code>	Build Configuration File
<code>package.xml</code>	Package Configuration File

The ROS build system 'catkin' uses the software tool CMake. Therefore, the build environment is described in the 'CMakeLists.txt' file in the package folder. This file configures executable file creation, dependency package build priority, link creation, and so on. Thus, UnitCall and BaseCall service files may be added in CMakeLists file 3.1.

```
## Generate services in the 'srv' folder
add_service_files(
  FILES
  UnitCall.srv
  BaseCall.srv
)
```

Figure 3.1: Srv files are added in CMakeLists file

This option will include the 'UnitCall.srv' and 'BaseCall.srv' when building the package, which will be used in the nodes.

3.1.2 UnitCall Service

UnitCall is called the service where Unit is the client that makes the request 3.2. The client node gets a number of values as input and connects through master with the server node. These values represent the current state of the Unit. Afterwards, client sends the values to server and waits for response.

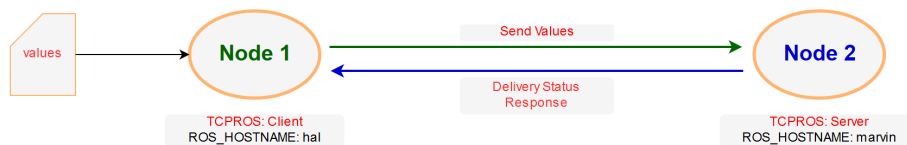


Figure 3.2: UnitCall Service

Part of the request code

```

rospy.wait_for_service('call')
try:
    call = rospy.ServiceProxy('call', Call)
    resp1 = call(x, y, k, a, j, r, q, t)
    return resp1.config
except rospy.ServiceException, e:
    print "Service call failed: %s"%e

rospy.wait_for_service('call')

```

This method blocks operation until the service named 'call' is available, preventing data loss. For example due to network connection failure. Afterwards a handle is created for calling the service.

```

call = rospy.ServiceProxy('call', Call)
resp1 = call(x, y, k, a, j, r, q, t)
return resp1.config

```

ServiceProxy class does the actual work of locating the server node, transmitting the request data and waiting for a response. The type of service is declared by UnitCall.srv file. The CallRequest object is sent to server as resp1 variable.

```

except rospy.ServiceException, e:

```

If the call fails by errors related to ROS Service communication, a rospy.ServiceException may be thrown.

Server node receives the values and responds back to client with an indication of success.

The server node initializes from the client's call. A server named 'call' is declared and all requests are passed to to `handle_call` function. This function includes instances of `CallRequest` and returns instances of `CallResponse` confirming that the values were received ('ok'). Unlike client node, the server node does not shut down after the operation is completed. `Rospy.spin()` asks ROS to wait for pending callbacks.

Part of the response code

```
rospy.init_node('call_server')
    s = rospy.Service('call', Call, handle_call)
    . . .
    rospy.spin()
def handle_call(req):
    . . .
    return ('ok')
```

Service file

UnitCall.srv file generates the `CallRequest` object that is sent to server. It also generates the `CallResponse` object and if the procedure is successful it returns a single string. Data before the dashes are the request elements and data after the dashes are the response elements as they are presenting in figure 3.3.

```
int64 a
int64 b
string c
string d
int64 e
int64 f
int64 g
string i
---
string config
```

Figure 3.3: Service file UnitCall

3.1.3 BaseCall Service

Basecall is called the service where Base node makes the request asking for the aforementioned values 3.4. It connects through master with the Unit node and waits for response. Afterwards, Unit sends the values to Base.

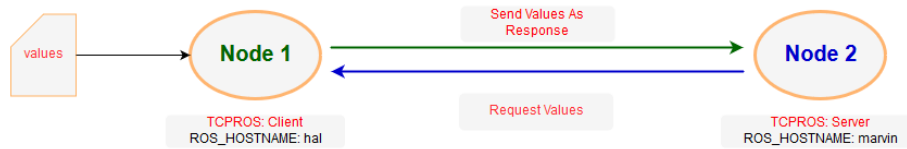


Figure 3.4: BaseCall Service

Request is sent through value 'a'. Then the values are sent to Unit in a list form, so in the BaseCall.srv that follows the response part is a string 3.5.

```
int64  a
---
string config
```

Figure 3.5: Service file BaseCall

UnitCall and BaseCall services work in the same way. This means they use the same main methods. But, they present differences related on which makes the call in each case. As the information is always transferred from Unit to Base, in the first case it passes as a request while in the second case it passes as response.

3.1.4 ROS Master

Initializing the whole ROS application it is necessary to start ROS master. One master is capable of managing both Unitcall and Basecall services.

Roscore is the node administrator and remains open during any ROS operation.

3.1.5 Network Setup in ROS

ROS system can support running even hundreds of nodes at the same time in multiple machines. So, in order for these nodes to be able to communicate with each other at any time, a network configuration is needed. The requirements are a bi-directional connectivity between all pairs of machines and on all ports and each machine must advertise itself by a name that all other machines can resolve [1].

When a ROS node advertises a topic, it provides a `hostname:port` combination (a URI) that other nodes will contact when they want to subscribe to that topic. It is important that the hostname that a node provides can be used by all other nodes to contact it. The ROS client libraries use the name that the machine reports to be its hostname. This is the name that is returned by the command `hostname`.

Defining the machines names

In this project, ROS system runs in two machines. In order to define the hostnames of the machines, the first step is to open the `/etc/hosts` file. The file includes IP addresses of the two machines and localhosts names. The next step is to add the IP address of the other node and also change their both names. Unit machine is called `hal` and Base machine is called `marvin`.

```
192.168.1.2  hal
192.168.1.1  marvin
```

The last step is to connect with the other machine through SSH and check the connectivity. The desired outcome of a sufficient connection is presented in figure 3.6.

```
ping hal
PING hal (192.168.1.2) 56(84) bytes of data.
64 bytes from hal (192.168.1.2): icmp_seq=1 ttl=64 time=0.016 ms
64 bytes from hal (192.168.1.2): icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from hal (192.168.1.2): icmp_seq=3 ttl=64 time=0.023 ms
64 bytes from hal (192.168.1.2): icmp_seq=4 ttl=64 time=0.049 ms
64 bytes from hal (192.168.1.2): icmp_seq=5 ttl=64 time=0.041 ms
64 bytes from hal (192.168.1.2): icmp_seq=6 ttl=64 time=0.053 ms
```

Figure 3.6: Successful connectivity with hal machine

Machine Configuration

Roscore should run in one of the two machines. So, it is necessary to set up an environment where all nodes can locate master. This is achieved by ROS environment variables, via `ROS_MASTER_URI`, because all nodes must be configured to use the same master. Roscore will run at Unit machine so it is needed to export the specific address: `export ROS_MASTER_URI=http://hal:11311`.

3.1.6 OpenCV

This application is designed to secure an area from human trespass. Therefore, the concept includes that a camera is mounted on Unit Point in order to track space violation by detecting human bodies.

Regarding the hardware in the simulation scenario, Unit machine uses its webcam in order to detect faces and eyes instead of human bodies.

As for the software part of the project, OpenCV library was chosen to make a Real-Time Face Detection. OpenCV comes with a number of built-in cascades for detecting everything from faces to eyes to hands to legs. Face Detection, using Haar feature-based cascade classifiers, is an effective and widely used object detection method with pretrained classifiers of human faces. Also, `cv_bridge` library from ROS was included in order to convert cv images to ROS messages and the opposite.

The main part of OpenCV code block uses LBPH (Local Binary Patterns Histograms) algorithm to detect faces. It labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

LBPH uses four parameters:

- Radius: the radius is used to build the circular local binary pattern and represents the radius around the central pixel
- Neighbors : the number of sample points to build the circular local binary pattern
- Grid X : the number of cells in the horizontal direction
- Grid Y : the number of cells in the vertical direction

Part of OpenCv Code Block

In this project the script detect.py was developed as a node which listens to a ROS image message topic, converts the images into a cv_image, makes face detection and displays the image using OpenCV. Then, the image is republished over ROS. This node is not a camera streamer, but uses cv_camera_node package to start the camera.

```
face_cascade =
↪ cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

def __init__(self):
    self.image_pub = rospy.Publisher("detected_face", Image)

    self.bridge = CvBridge()
    self.image_sub =
↪ rospy.Subscriber("/cv_camera/image_raw", Image, self.callback)

def callback(self, data):

    try:
        cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
        print(e)

    gray = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
    faces = detector.detectMultiScale(gray, 1.3, 5)
```

```

    print(len(faces))
    . . .

    for (x,y,w,h) in faces:
        cv2.rectangle(cv_image,(x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow("Faces found", cv_image)
cv2.waitKey(1)

try:
    self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image,
        ↪ "bgr8"))
except CvBridgeError as e:
    print(e)

rospy.init_node('face_detection', anonymous=True)

face_cascade =
    ↪ cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

```

This line loads the face cascade into memory. The cascade is an XML file that contains the data to detect faces.

```

self.image_pub = rospy.Publisher("detected_face",Image)
self.bridge = CvBridge()
self.image_sub =
    ↪ rospy.Subscriber("/cv_camera/image_raw",Image,self.callback)

```

These lines create a cv_bridge object, a publisher which will republish the images as ROS messages and a subscriber. The detect node subscribes to /cv_camera/image_raw topic from cv_camera_node to use frames from camera.

```

cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")

```

This line converts a ROS image message into a cv image through module cv_bridge.CvBridge. The input is the image message, as well as an optional

encoding. The encoding refers to the destination cv image. The "bgr8" refers to a color image and it is more expected by most OpenCV functions.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

The captured frame is converted to gray scale before using the face detector.

```
faces = detector.detectMultiScale(gray, 1.3, 5)
```

```
scaleFactor=1.3
```

```
minNeighbors=5
```

The above line applies the face detector to detect faces in the captured frame. This function detects the actual face and is the key part of the code. The options inside the brackets describe the following:

- The detectMultiScale function is a general function that detects objects. Since it is called on the face cascade, it detects faces.
- The first option is the grayscale image.
- The second is the scaleFactor. Since some faces may be closer to the camera, they would appear bigger than the faces in the back. The scale factor compensates for this.
- The detection algorithm uses a moving window to detect objects. MinNeighbors defines how many objects are detected near the current one before it declares the face found. MinSize, meanwhile, gives the size of each window.

```
print(len(faces))
```

The len(faces) function returns the number of faces in a detected frame. If a face is detected the algorithm returns 1, otherwise it returns 0. A result is printed for every captured frame 3.7.

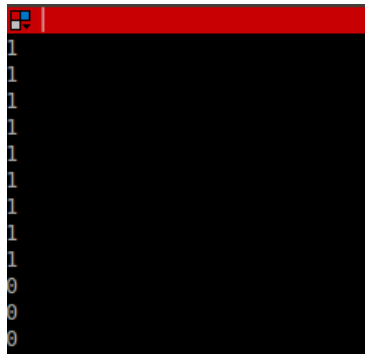


Figure 3.7: Result for every captured frame in terminal window

```
for (x,y,w,h) in faces:  
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
```

The function returns a list of rectangles in which it believes it found a face. This function returns four values: the x and y location of the rectangle, and the rectangle's width and height (w , h). These values are needed in order to draw a rectangle using the built-in `rectangle()` function.

The options inside the brackets describe the following:

- The first argument is the input frame
- The second is the x,y coordinate of the face
- Then is the height and weight
- After that the color of the line is specified which is in the form of (blue,green,red) with adjustable values, range is 0-255. In this project is used green line
- The last argument is the line thickness

```
cv2.imshow("Faces found", cv_image)  
cv2.waitKey(0)
```

These lines are displaying the frame and wait for the user to press a key respectively.

```
self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
```


This line converts a cv image into a ROS image message through CvBridge and then publishes it over ROS.

```
rospy.init_node('face_detection', anonymous=True)
```

This lines creates a unique node with name 'face_detection'.

3.1.7 Sql

Daily, a large amount of data that is being transferred from Unit Point to Base Point. A backup history of this data is required once in every 24 hours in order to compare values between days or even between weeks.

Therefore, a collection of incoming information requires organization which is achieved by creating a database. Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

The software library that was used to provide a relational database management system is Sqlite. The python code block that follows, creates a table named "UnitData" in order to store and classify the incoming data from Unit point.

Part of Database Code Block

```
conn = sqlite3.connect('test.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS UnitData
(sent_date TEXT,
sent_time TEXT,
received_date TEXT,
received_time TEXT,
Camera TEXT,
Battery TEXT,
Environment Temperature TEXT,
Device Temperature TEXT,
Barometer TEXT,
```

```
GPSlat TEXT,  
GPSlon TEXT);''')
```

```
cursor.execute("INSERT INTO UnitData  
→ VALUES(?,?,?,?,?,?,?,?,?,?,?)", (req.i[0:10], req.i[11:],  
→ t[0:10], t[11:], cam, req.a, req.b, req.g, req.f, req.c,  
→ req.d))
```

```
conn = sqlite3.connect('test.db')
```

To use Sqlite3, a connection object should be created that represents the database. The data will be stored in the test.db file.

```
cursor = conn.cursor()
```

The above line creates a connection object then uses it to instantiate a cursor object. The cursor object is used to execute SQL statements on the Sqlite database.

```
cursor.execute('''CREATE TABLE IF NOT EXISTS UnitData  
(sent_date TEXT,  
sent_time TEXT,  
received_date TEXT,  
received_time TEXT,  
Camera TEXT,  
Battery TEXT,  
Environment Temperature TEXT,  
Device Temperature TEXT,  
Barometer TEXT,  
GPSlat TEXT,  
GPSlon TEXT);''')
```

The execute() method of the created cursor is called to perform SQL commands. The "IF NOT EXISTS" statement is used to avoid overwriting an existing table.

This table consists of 11 columns, each one of them describes the name and type of the incoming data. Type "INT" is an integer data type. "REAL" is a floating point value and "TEXT" is a text string.

```
cursor.execute("INSERT INTO UnitData
→ VALUES(?,?,?,?,?,?,?,?,?,?)", (req.i[0:10], req.i[11:], t,
→ req.a, req.b, req.c, req.d, req.e, req.f, req.g))
```

The values are inserted in table "UnitData", each one under the corresponding name.

3.1.8 Html

Unit point includes a group of sensors, such as camera, GPS, thermometer, barometer which collect information about Unit Point's state. This information (data) is sent through Sender-Receiver method using ROS to Base Point, in order to be evaluated by a human who should have the ability to easily access, manage and update data at anytime. Data might contain urgent information about Unit's safety, for example camera's battery usage or the exact time of a trespass occurrence, and therefore they need to appear as warnings.

A web application using Html offers a simple way to display and control data. Values are retrieved from a database and they appear in a html table in a web page. Also, warnings could be pop up windows in front of the screen.

Http

HTTP is the protocol for websites. It is designed to interact and communicate with computers and servers. HTTP works as a request-response protocol between a client and server. When a name of a website in the address bar of a browser is typed, what happens is that an HTTP request has been sent to a server. That server receives the request and needs to figure how to interpret that request. Next, sends back an HTTP response that contains the information that the web browser receives and then, it displays what is asked for on a page in the browser.

A web browser may be the client and ROS application may be the server. This

enables user to submit an HTTP request from the Base point and through ROS communication get a response from Unit point's status.

For this web application, a combination of Python and HTML was used. Python was executed on the server side while HTML was downloaded to the client and run by the web browser. Python was used to generate Html and the latter for constructing the layout of the application's interface and finally to display its elements on screen with the Css (Cascading Style Sheets) language.

Flask Application

Flask is a Python framework for creating web applications. It makes the designing of a web application simpler as it focuses on what the users are requesting and what sort of response to give back. Also, it makes development faster by offering code for all sorts of processes like database interaction or file activity.

The app.py file will contain the main code that will be executed by the Python interpreter to run the Flask web application. The templates directory is the directory in which Flask will look for static HTML files for rendering in the web browser. Templates folder contains two html files that will be used in this part of the code. Furthermore, the static directory is created as well which contains a css file which demonstrates Html's document style. The directory tree that follow describes this concept.

```
1st_flask_app-1 /
  app.py
  templates/
    print_items.html
    blank.html
  static/
    style.css
```

Before start analyzing the contents of both python script app and html templates, it is important to mention that as the one is depending on the other in order to run, a step-by-step code presentation will take place.

```
from flask import Flask, render_template, request
```

```
import Tkinter as tk
import tkMessageBox

app = Flask(__name__)
@app.route('/')
@app.route('/print_items', methods=["GET", "POST"])

def print_items():
    ...
    tkMessageBox.showwarning('Base', 'Alert!')
    ...
    return render_template('print_items.html', items=items)

@app.route('/blank', methods=["GET", "POST"])
def blank2():
    ...
    return render_template('blank.html', items=items)
```

In order to run the application as a single module, a new Flask instance is initialized with the argument `__name__` to let Flask know that it can find the HTML template folder (`templates`) in the same directory where it is located.

The route decorator (`@app.route('/')`) is used to specify the URL that should trigger the execution of the `print_items` and `blank` functions, therefore it binds the functions with the URL. These three functions simply render the HTML files `print_items.html` and `blank.html`, by using the `render_template()` method. This means that it actually displays their content in the user's browser. Inside the brackets is provided the name of the html file and the variables that are going to pass to the template engine as keyword arguments. Variable "items" from the first two functions contains data from the database "UnitData" which will pass to `print_items.html` and `blank.html`. `Print_items` function also contains a `MessageBox()` method that pops up a message window in the rendered `print_items` Html page under conditions.

The HTTP method knows different methods for accessing URLs. The HTTP methods 'GET', 'POST' are used to succeed HTTP 'client-server' interaction. By default, a route only answers to GET requests, but that can be changed by

providing the `methods` argument to the `route()` decorator. The POST method is used for the `'print_items'` and `'blank'` templates in order to transport the form data to the server in the message body. POST is better for submitting secure data as they do not appear in the address bar like GET method.

These two functions display their HTML content in the user browser, in two different web pages, after the user (human) make an action such as clicking a button. Each one of the functions have conditions that handle the incoming data from the database and depend on what kind of information the user would request. Specifically, in the HTML code below, four buttons are created in order to make options for the user to choose what kind of information he wants to receive. These kind of information are going to be explained thoroughly in the section below.

The `app.py` script runs locally by executing the following command from the app's main directory and the user can view the result in his web browser at `http://127.0.0.1:5000/`. The result is the content of the `print_items` HTML file. The user sees a table in which its rows are filled with data according to some conditions and three buttons ready for submission. When the user decides to click one of them, their content is rendered in a new page or fills the table with updated data.

```
$ python app.py
```

At first the `'print_items'` template loads the CSS file which creates a scrolling table where the data will be displayed. After the table has been created with the corresponding name values as rows, they are filled with data from the variable `"items"` of the `python app.py` script. The HTML also makes a refresh after 5 seconds to update the table in case new data came in. The lines that follow describe the creation of four submit buttons. The HTML `<form>` element defines a form that is used to collect user input. After the user click the buttons, named `"new message"` or `"photo"`, the form data is sent to the page `"/print_items"` and the table data are updated.

```
<form>
  <form method=post action="/print_items">
    <input type=submit value='new message' name='action'>
  </form>
```

The submit button "show alerts", was created in order to open a new page when the user chooses to click it. The variable "items", as in print_items case, loads data from the database "UnitData" and display them in a scrolling table but this time with specific rows and columns. Next, if the user chooses to click the button "photo", a window is opened containing a photo. Alternative, if the user chooses to click the button "video", a window is opened containing a real time video.

```
<form>
  <form method=post action="/print_items">
    <input type=submit value='show alerts' name='action'>
  </form>
```

```
<form>
  <form method=post action="/print_items">
    <input type=submit value='photo' name='action'>
  </form>
```

```
<form>
  <form method=post action="/print_items">
    <input type=submit value='video' name='action'>
  </form>
```

3.2 Application Description

As mentioned previously, the application's actual implementation took place in the National University's Marine Laboratory using two machines. Each one of them represents the remote locations (Unit/Base) where telemetry was implemented. So far, application's main cause has been stated, but not approached from user's side.

The user is able to check several values which represent the state of the Unit. These values refer to battery level, device's and environment's temperature, device's location and human detection, as well as date and time information. For that reason, the user has a platform in his disposal 3.8, in which all the messages, that are being sent from Unit, are displayed. Incoming messages are

inserted into a table, with every row constitutes of a message. The platform is being refreshed periodically and messages are sorted by the time of shipment.

Base											
Sent_date	Sent_time	Received_date	Received_time	Camera	Battery	Env Temp	Dev Temp	Barometer	GPSlat	GPSlon	
2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:30.303438	2019-06-03	22:26:30.614738	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:19.601162	2019-06-03	22:26:19.644960	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:18.225730	2019-06-03	22:26:19.074889	Alert	50	15	60	2	37.9842	23.7353	



Figure 3.8: Application's main platform

Among the information a new message holds, an indicator shows if Unit's camera detected a face or not. In case face detection is positive, user receives an alert message in the form of pop up window in the platform.

The user can seclude alerted messages in order to handle more efficient a large amount of incoming messages during the day. This can be achieved by clicking the option "show alerts".

Every new message is added in the table after an automatic repeatable procedure. This procedure is held by UnitCall Service which has described in section 3.1.2. Besides that, the user can request for Unit's status at any time. This is achieved with options "new message" and "photo". Both options return a new message with the only deference that "photo" requests additionally for an instant image. This image displays the area around Unit depending on web camera's range. In this way, the user can have a visual perspective in case of a trespass, similar to option "video" where the user can examine real time the area through live cast video. These two options connected to "new message" and "photo" buttons activate BaseCall Service 3.1.3.

In conclusion, this application uses two different services in order for the messages to be transmitted from Unit to Base 3.9 3.15.

3.2.1 UnitCall Development

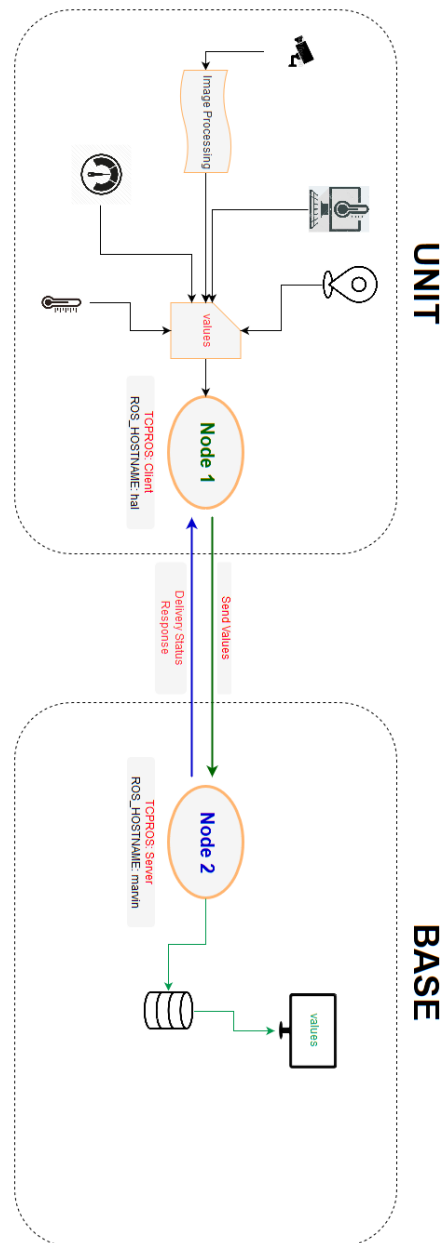


Figure 3.9: UnitCall Service

In this project, Unit gathers information from the following sources.

Camera

Unit uses two different nodes to start image processing. As mentioned in 3.1.6, `cv_camera_node` runs in order to start the camera and publish images in `/cv_camera/image_raw` topic. Afterwards, the node `detect.py` subscribes to `/cv_camera/image_raw` to use the images and then starts the processing part. Lastly, the images after image processing are republished in "detected_face" topic 3.10.

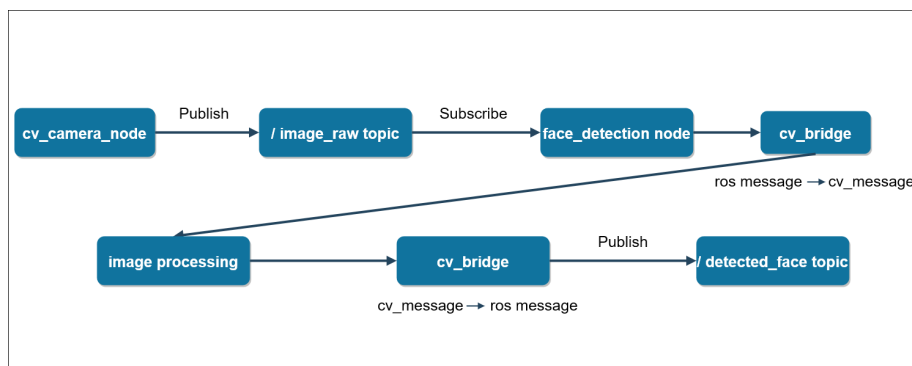


Figure 3.10: Unit's camera process with nodes topics and `cv_bridge`

PC's web cam is used for image processing. As mentioned in 3.1.6, OpenCv uses a function to return the number of detected faces. These values represent integers, 1 for detected face and 0 for non detected. Then, these values are extracted into a txt file, called `camdata.py`, along with a string timestamp. Additionally, in some cases values become -1 e.g. if OpenCv stops working 3.11.

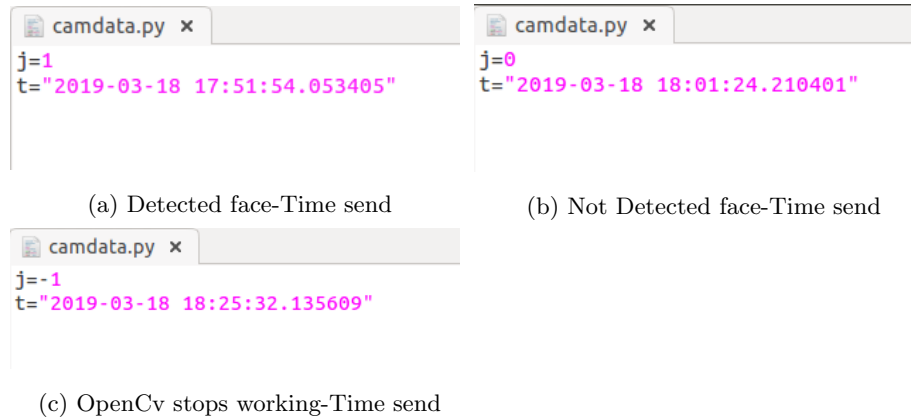


Figure 3.11: Camdata Values

Time sent Timestamp regarding to when the message is sent (date and time) from the Unit point. It is stored in camdata.py file.

GPS Unit's position is received from the website <https://ipstack.com/> as geographical longitude and latitude.

Other Sensors Battery's levels, environment's and device's temperature, as well as barometer's pressure are defined by random default values in a txt file, called values.py 3.12.

```

values.py x
#! /usr/bin/env python

import sys

Bat = 50
EnvT = 15
DevT = 60
Bar = 2

```

Figure 3.12: Default Sensor Values

These python files are imported into application's basic executable program called "call_list.py". The individual variables which are contained in these files, consist of the message that is going to be sent from Unit to Base. The variables are passing as arguments in ROS command for running service client as

it showing in figure 3.13.

```
os.system('roslaunch application_unit call_client.py %s %s %s %s %s %s %s' %(x, y, lat, lon, j, r, q, t))
```

Figure 3.13: ROS Command to initiate UnitCall Service

Call_list is developed to run "call_client" service automatically under two conditions. These conditions determine when the message is going to be sent.

Standard message

Normally, a message is sent to "Base" every 5 minutes. Along with the status of "Unit", the message also contains a value (j=1 or j=0) that indicates a trespass or not.

Alert message

Every time there is a new human trespass in the area and so the result from OpenCv is j=1, "Unit" informs the "Base" with an alert message. OpenCv generates a value (1 or 0) for every captured frame meaning several values per second. This is a large and unnecessary amount of information as "Base" needs to be informed only once by the time and duration of the event. Therefore, call_list and detect were implemented to send the message once when j=1 for the first time, implying there was a trespass, and then send the message again when j=0 for the first time as well, implying human has left the area.

The messages are sent from "Unit" to "Base" using ROS communication system. Call_Server is developed to receive messages coming from "Unit" and then register the individual values in a table in database. Client-Server communication description can be found in detail in 2.1.7.

Table UnitData has 11 titles in columns, each one of them represents the names of the incoming variables 3.14. Every new message is a new row in the table and their values are placed in the corresponding name.

	sent_date	sent_time	received_date	received_time	Camera	Battery	Environment	Device	Barometer	GPSlat	GPSlon
1	2019-06-03	22:26:18.225730	2019-06-03	22:26:19.074889	Alert	50	15	60	2	37.9842	23.73
2	2019-06-03	22:26:19.601162	2019-06-03	22:26:19.644960	Alert	50	15	60	2	37.9842	23.73
3	2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.73
4	2019-06-03	22:26:30.303438	2019-06-03	22:26:30.614738	OK	50	15	60	2	37.9842	23.73
5	2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.73
6	2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.73
7	2019-06-03	22:42:33.347526	2019-06-03	22:42:33.433771	OK	50	15	60	2	37.9842	23.73

Figure 3.14: Database UnitData

App.py file opens a HTML page in which the content of the "Unitdata" database table is displayed. The function `print_items` makes a connection with the database and then fetches all rows to a HTML table. The function is triggered every 5sec using refresh module. In case UnitData has been filled with a new message, the table Base is updated with a new row. In addition, in case of an alert message, a pop up window appears in order to be noticed by the control staff.

```

conn = sqlite3.connect('~/.Unitdata.db')
cursor = conn.cursor()
cursor.execute('SELECT * FROM COMPANY ORDER BY recieved_time
→ DESC')
items=cursor.fetchall()
cursor.execute('SELECT * FROM COMPANY ORDER BY sent_time DESC
→ LIMIT 1')
k=cursor.fetchone()[7]
cursor.execute('SELECT * FROM COMPANY ORDER BY sent_time DESC
→ LIMIT 1')
z=cursor.fetchone()[2]
global x
if k==1 and z!=x:
    root=tk.Tk()
    root.withdraw()
    tkinter.messagebox.showwarning('Base', 'Alert!')
    x=z
return render_template('print_items.html', items=items)

```

3.2.2 BaseCall Development

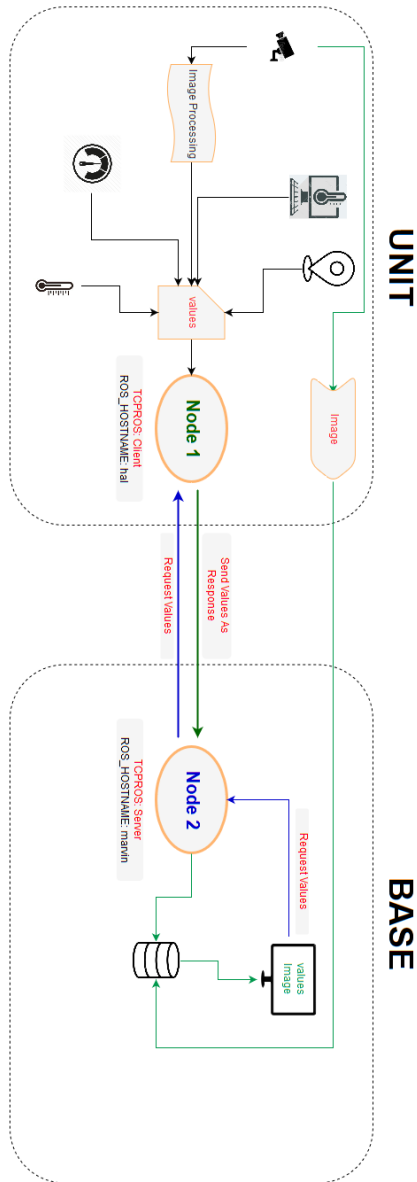


Figure 3.15: BaseCall Service

Standard messages can be requested any time from user in "Base". When "new message" or "photo" options are pressed, BaseCall service is activated from app.py file, in order to receive the standard message instantly.

```

if request.method=='POST' and request.form['action'] ==
→ 'NewMessage':
    os.system('python ~/request_server.py 0')
elif request.method=='POST' and request.form['action'] ==
→ 'Photo':
    os.system('python ~/request_server.py 1')
elif request.method=='POST' and request.form['action'] ==
→ 'Video':
    os.system('rqt_image_view')

```

In case the "new message" button is pressed, "request_server" takes "0" as argument (a=0). Alternatively, if "photo" is pressed, request_server" takes "1" as argument (a=1). In both cases the value is sent to "request_client" which instantly updates the sensors' data and responds to "Base".

```

reload(values)
from values import x, y, r, q
reload(camdata)
from camdata import j, t
reload(gps)
from gps import lat, lon
l=[x,y,lat,lon,j,r,q,t]
return "%s"%(l)

```

However, in "photo" case, an image is sent back to "Base". This is achieved with using the image_saver ROS package 2.1.9. The node image_saver runs in Unit with additionally parameters to filter the script and waits until a call is made in order to save an image 3.16.

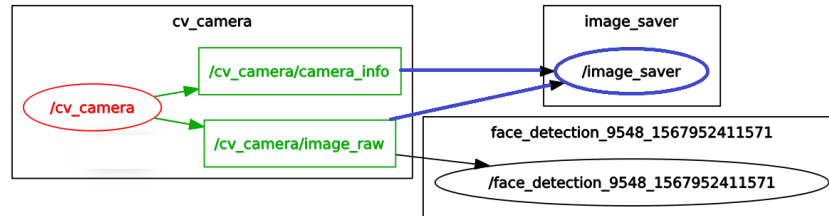


Figure 3.16: Image_saver listens to cv_camera_node topic and saves an image until a service call is made

```
$ rosrun image_view image_saver image:=/cv_camera/image_raw
↪ _save_all_image:=false _filename_format:=image.jpg
↪ __name:=image_saver

if req.a==1:
    os.system('rosservice call /image_saver/save')
    os.system('scp ~/image.jpg
↪ base@marvin:~/web_application/static/image')
```

Then every time the option "photo" is pressed and therefore $a=1$, "request_client" activates "image_saver" to capture an image and store it in "Unit" machine's hard drive. Then the image is transferred to "Base" machine through Secure Shell (SSH).

When the image is saved to "Base", "app.py" displays the image in a window using again the Tk() module as message pop up window.

```
img = ImageTk.PhotoImage(Image.open(path))
panel = Label(root, image = img)
panel.pack(side = "bottom", fill = "both", expand = "yes")
```

The standard message is stored in "Unitdata" table with the same procedure as "Unitcall" service.

Lastly, the button "video" opens ROS plugin `image_view`. This way user chooses visualization of the topics which are running in Unit. This is accomplished due to export `ROS_MASTER_URI` where all nodes in "Base" can locate and use nodes running in "Unit". According to these facts, in `image_view` two topics are available, "detected_face" and `"/cv_camera/_image_raw"`. Both display the area around Unit with the only difference that `"/detected_face"` has undergone image processing.

A flow diagram of the process is presented in 3.17.

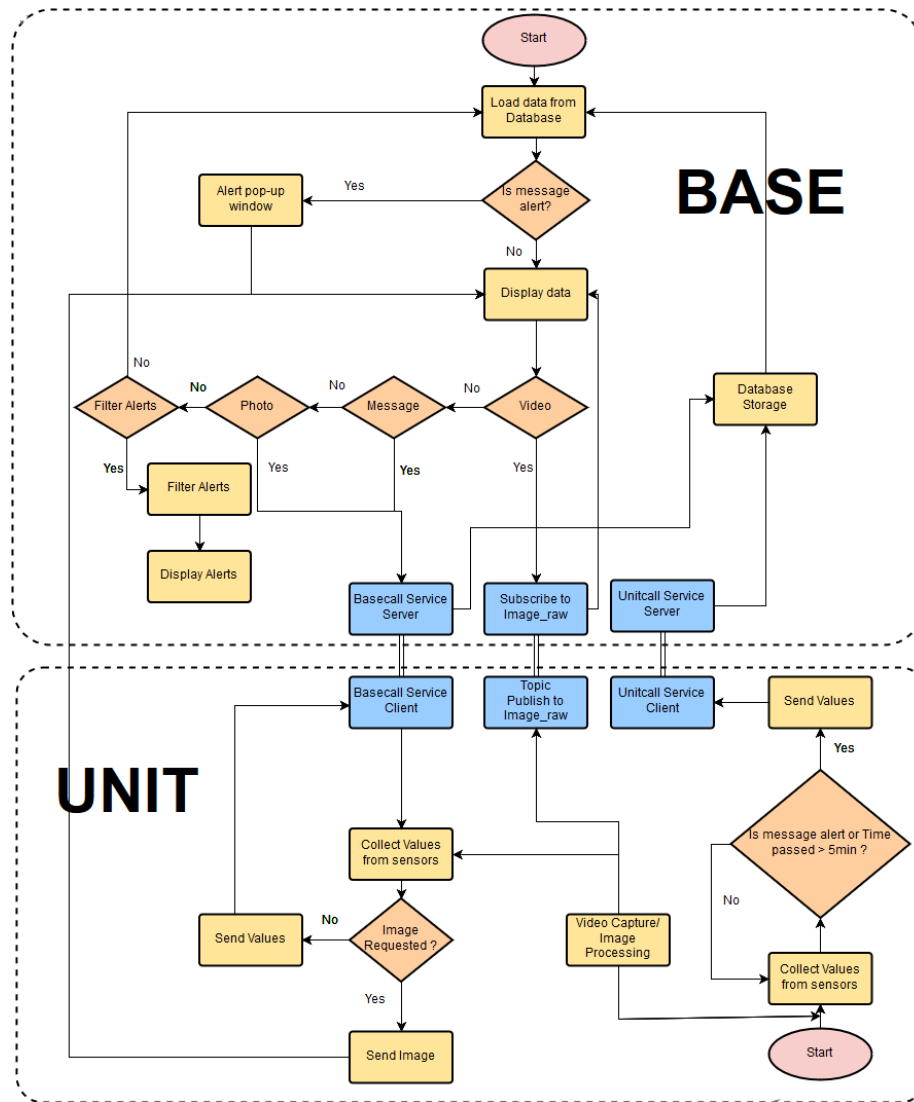


Figure 3.17: Flow diagram

Chapter 4

Results

This chapter describes the functionality of the application, thus it presents screen screenshots presentation to explain step by step the final application design.

4.0.1 Application prerequisites

In order to be able to begin with the installation and running the application, following prerequisites must be installed on the Unit machine:

- Ubuntu 14.04
- Python 2.7
- ROS Indigo
- OpenCV 2.4
- cv_camera_node
- image_saver

And the following prerequisites must be installed on the Base machine:

- Ubuntu 14.04

- Python 2.7
- ROS Indigo
- Sqlite

4.0.2 Application Startup

Base startup screen

The Base table is the first screen that user faces after entering the application URL. The platform contains four available buttons, "new message", " show alerts" , "photo", "video".

Base										
Sent_date	Sent_time	Received_date	Received_time	Camera	Battery	Env Temp	Dev Temp	Barometer	GPSlat	GPSlon
2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:30.303438	2019-06-03	22:26:30.614738	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:19.601162	2019-06-03	22:26:19.644960	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:18.225730	2019-06-03	22:26:19.074889	Alert	50	15	60	2	37.9842	23.7353

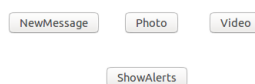
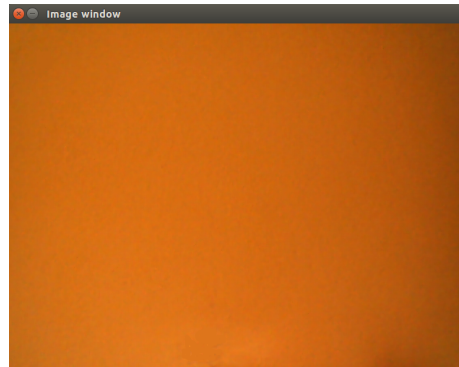


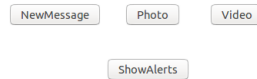
Figure 4.1: Startup Screen-Base

The table 4.1 displays the Unit's status. The Base table is auto-refreshed periodically in order to receive the updated status. User can also manually refresh the HTML page at any time. As long as there is no trespass, the value 'ok' is presented along with the rest of the Unit's status 4.2.



(a) Window displaying the surveillance area

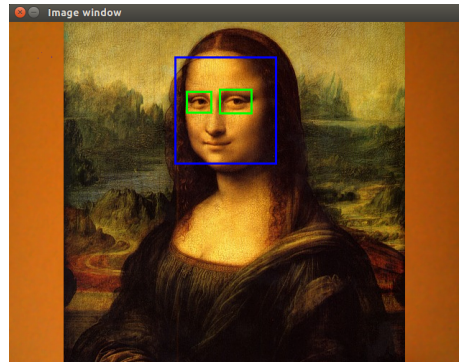
Base											
Sent_date	Sent_time	Received_date	Received_time	Camera	Battery	Env Temp	Dev Temp	Barometer	GPSlat	GPSlon	
2019-06-03	22:43:38.335579	2019-06-03	22:43:39.559904	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:33.043749	2019-06-03	22:43:34.856873	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:28.071112	2019-06-03	22:43:28.473393	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:27.288888	2019-06-03	22:43:27.605727	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:22.221273	2019-06-03	22:43:23.056897	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:19.100056	2019-06-03	22:43:19.400503	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:17.273052	2019-06-03	22:43:17.593915	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:12.005014	2019-06-03	22:43:12.218131	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:06.357653	2019-06-03	22:43:06.672648	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:01.287360	2019-06-03	22:43:01.535232	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:55.432524	2019-06-03	22:42:55.694892	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:50.204881	2019-06-03	22:42:50.368554	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:44.000425	2019-06-03	22:42:44.817098	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:41.397808	2019-06-03	22:42:41.693021	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:39.101399	2019-06-03	22:42:39.917606	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:38.279178	2019-06-03	22:42:38.623173	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:35.163988	2019-06-03	22:42:35.506916	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:33.232526	2019-06-03	22:42:33.343771	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:30.303438	2019-06-03	22:26:30.614738	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.7353	



(b) Base table

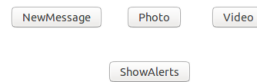
Figure 4.2: Unit's status without trespass

When Unit detects a threat, a message is sent instantly to Base and the alert window pops up 4.3.



(a) Window displaying the surveillance area

Base											
Sent_date	Sent_time	Received_date	Received_time	Camera	Battery	Env Temp	Dev Temp	Barometer	GPSlat	GPSlon	
2019-06-03	22:58:53.208049	2019-06-03	22:58:53.42549	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:58:52.495471	2019-06-03	22:58:52.812359	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:38.335579	2019-06-03	22:43:39.559904	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:33.043749	2019-06-03	22:43:34.856873	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:28.071112	2019-06-03	22:43:28.473393	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:27.288888	2019-06-03	22:43:27.605727	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:22.221273	2019-06-03	22:43:23.056897	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:19.100056	2019-06-03	22:43:19.400503	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:17.273052	2019-06-03	22:43:17.593915	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:12.105014	2019-06-03	22:43:12.218131	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:06.357653	2019-06-03	22:43:06.6726	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:01.187360	2019-06-03	22:43:01.2552	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:55.432524	2019-06-03	22:42:55.6948	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:50.264881	2019-06-03	22:42:50.3085	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:44.000425	2019-06-03	22:42:44.8170	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:41.397808	2019-06-03	22:42:41.6930	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:39.091399	2019-06-03	22:42:39.117606	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:38.279178	2019-06-03	22:42:38.623173	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:35.163988	2019-06-03	22:42:35.506916	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:33.242526	2019-06-03	22:42:33.333771	OK	50	15	60	2	37.9842	23.7353	
2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:36.176275	2019-06-03	22:26:36.257629	OK	50	15	60	2	37.9842	23.7353	



(b) Base table with alert message

Figure 4.3: Unit's status with trespass

After clicking the 'ok' button, the pop up window closes, and the user can examine the alert message 4.4.

Base										
Sent date	Sent time	Received date	Received time	Camera	Battery	Env. Temp	Dev Temp	Barometer	GPSlat	GPSlon
2019-06-03	22:58:53.208049	2019-06-03	22:58:53.42549	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:58:52.495471	2019-06-03	22:58:52.812359	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:38.335579	2019-06-03	22:43:39.559904	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:33.043749	2019-06-03	22:43:34.856873	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:28.071112	2019-06-03	22:43:28.473393	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:27.288888	2019-06-03	22:43:27.605727	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:22.221273	2019-06-03	22:43:23.056897	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:19.100056	2019-06-03	22:43:19.400503	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:17.273052	2019-06-03	22:43:17.593915	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:12.005014	2019-06-03	22:43:12.218131	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:06.357653	2019-06-03	22:43:06.672648	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:01.287360	2019-06-03	22:43:01.535232	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:55.432524	2019-06-03	22:42:55.694892	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:50.204881	2019-06-03	22:42:50.368554	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:44.000425	2019-06-03	22:42:44.817098	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:41.397808	2019-06-03	22:42:41.693021	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:39.101399	2019-06-03	22:42:39.917606	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:38.279178	2019-06-03	22:42:38.623173	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:35.163988	2019-06-03	22:42:35.506916	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:33.232526	2019-06-03	22:42:33.343771	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.7353

Figure 4.4: Alert message

new message

By clicking the 'new message' button, the user can request a new message from the Unit. After a few seconds, the message is displayed on the screen 4.5.

Base										
Sent date	Sent time	Received date	Received time	Camera	Battery	Env. Temp	Dev Temp	Barometer	GPSlat	GPSlon
2019-06-03	22:43:38.335579	2019-06-03	22:43:39.559904	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:33.043749	2019-06-03	22:43:34.856873	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:28.071112	2019-06-03	22:43:28.473393	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:27.288888	2019-06-03	22:43:27.605727	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:22.221273	2019-06-03	22:43:23.056897	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:19.100056	2019-06-03	22:43:19.400503	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:17.273052	2019-06-03	22:43:17.593915	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:12.005014	2019-06-03	22:43:12.218131	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:06.357653	2019-06-03	22:43:06.672648	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:43:01.287360	2019-06-03	22:43:01.535232	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:55.432524	2019-06-03	22:42:55.694892	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:50.204881	2019-06-03	22:42:50.368554	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:44.000425	2019-06-03	22:42:44.817098	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:41.397808	2019-06-03	22:42:41.693021	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:39.101399	2019-06-03	22:42:39.917606	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:38.279178	2019-06-03	22:42:38.623173	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:35.163988	2019-06-03	22:42:35.506916	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:42:33.232526	2019-06-03	22:42:33.343771	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:36.276275	2019-06-03	22:26:36.517629	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:30.303438	2019-06-03	22:26:30.614738	OK	50	15	60	2	37.9842	23.7353
2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.7353

Figure 4.5: New message is added on table Base

show alerts

By clicking the 'show alerts' button, the user can seclude rows that containing

only alert messages 4.6.

Base											
Sent_date	Sent_time	Received_date	Received_time	Camera	Battery	Env Temp	Dev Temp	Barometer	GPSlat	GPSlon	
2019-06-03	22:26:18.225730	2019-06-03	22:26:19.074889	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:19.601162	2019-06-03	22:26:19.644960	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:26:25.233446	2019-06-03	22:26:25.411185	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:41:51.479847	2019-06-03	22:42:32.836698	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:35.163988	2019-06-03	22:42:35.506916	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:38.279178	2019-06-03	22:42:38.623173	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:39.101399	2019-06-03	22:42:39.917606	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:41.397808	2019-06-03	22:42:41.693021	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:42:44.000425	2019-06-03	22:42:44.817098	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:19.100056	2019-06-03	22:43:19.400503	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:22.221273	2019-06-03	22:43:23.056897	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:27.288888	2019-06-03	22:43:27.605727	Alert	50	15	60	2	37.9842	23.7353	
2019-06-03	22:43:28.071112	2019-06-03	22:43:28.473393	Alert	50	15	60	2	37.9842	23.7353	

Figure 4.6: Secluded alert messages

photo

In case a visual contact is required, the 'photo' button requests, along with a new message, an image from Unit. Then, the image is displayed in a pop-up window 4.7.

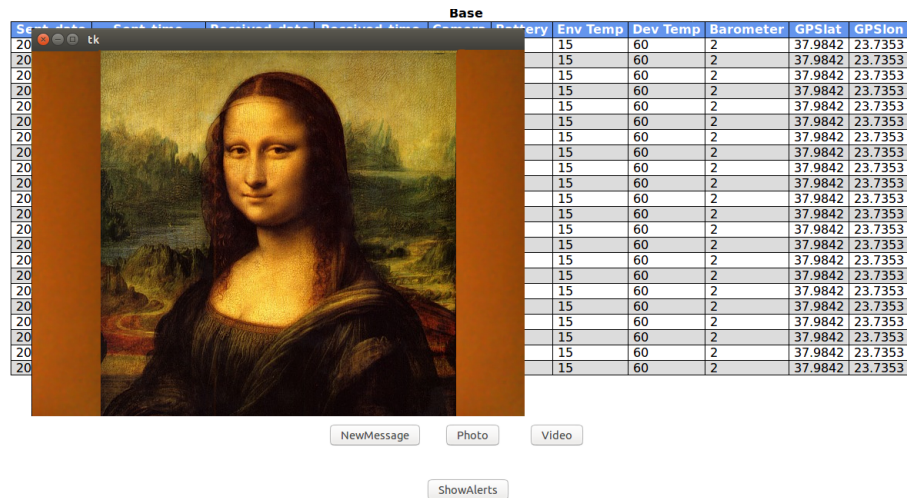
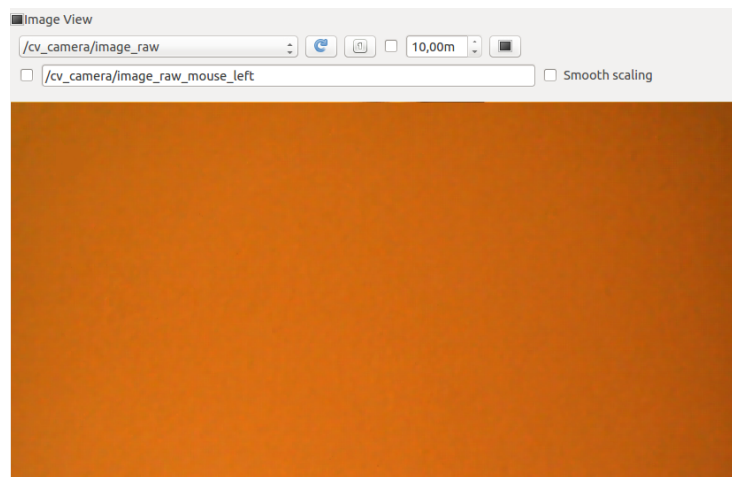


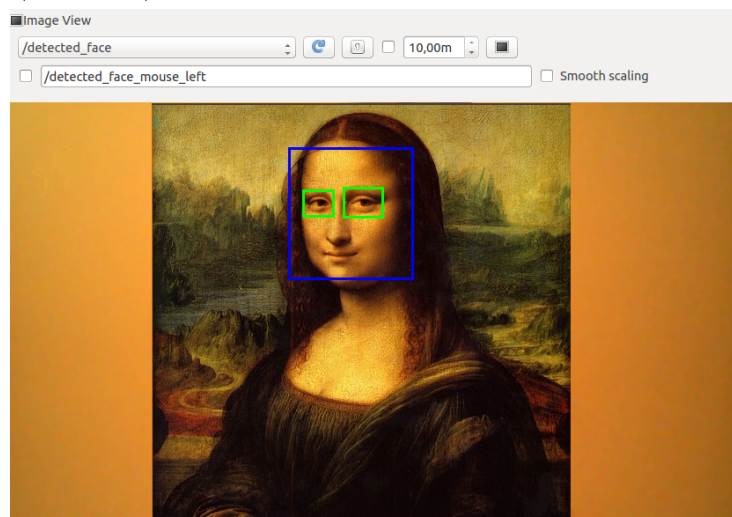
Figure 4.7: Photo displayed in Base screen

video

There are cases where the user not only wants to use a single image to examine the Unit's territory, but also use an option of a real time video streaming from camera. This is achieved with "video" button 4.8.



(a) ROS rqt_image_view displaying the surveillance area-Topic "/cv_camera/image_raw" selected



(b) ROS rqt_image_view displaying the surveillance area-Topic "/detected_face" selected

Figure 4.8: Video area surveillance

Chapter 5

Conclusions and Future Work

5.0.1 Conclusions

The aim of this master thesis was to design and develop an application that will have the merit to operate under a ROS software environment. In the project the idea of constructing a complete security system was implemented in order to inspect and secure a distributed area from possible trespass. ROS service was an ideal method for communication and data transferring as well as ROS topics and plugins for integrating with sensors. The individual software parts were selected in order to be simple, efficient and integrate with each other's environment. Also, taking into consideration that such a system requires nothing more than two machines containing average processing power with a typical web camera, the applicability of such a system is widened due to the low cost.

The project was tested under laboratory circumstances with two machines as Unit and Base point. The system demonstrated its ability to provide a communication model where messages are successfully sent either automatic or under request. Among that, the application detects faces and eyes with acceptable accuracy, while sends data and images to remote machine where they are stored and then displayed on screen. The communication was tested under local network and the messages as well as the images were being sent at acceptable speed.

In case of possible network failure, the application is temporarily blocking the operation in order to prevent message loss. The messages are successfully sent after the communication is restored. Finally, real time video adds a visual perspective in user's possession. Though, in remote connections the streaming seems to be a bit slow.

5.0.2 Future Work

Construction of a Unit Device-Real Sensors in Unit Device

When the project becomes realisable real sensors are going to be mounted in Unit Device, transmitting all the necessary information as messages. Also, the web camera will be replaced with stereo camera which will provide higher resolution and wider range. These new hardware functionalities are proposed to integrate and upgrade the software development of this project.

ROS Improvements

Due to the enhancement with more sensors in Unit device, more nodes need to be implemented in order to control each sensor and its messages. A new ROS service or a topic node for every new driver sensor should be developed, depending on what kind of communication it is needed to achieved.

OpenCv Improvements

In this project OpenCv detects faces and eyes with the pre-trained haarcascades. In the real implementation camera should recognize humans as divers, since the Unit Device is placed underwater.

Design Improvements

Application's platform design was created in order to provide the basic features for a user. It is needed further design improvements to become even more user-friendly and also even support an extra implementation for a touch screen mode.

Network Investigation

The tests in University laboratory used LAN Wi-Fi connection between the two machines. In further implementation it is proposed internet broadcast from satellite which will eliminate the costs in sending messages and the possibility

of connection loss.

Network-Application Security

It is recommended in both PC hard drives, to be provided with passwords on restart and also, network security methods. Furthermore, application's platform should be enhanced with "login" and "password" section in order to provide permission only to qualified staff.

Multi-threading

The project required concurrent process of data files in different time. This fact is controlled more efficient without limitations and less future bugs with multi-threading nodes.

Bibliography

- [1] ROS.org <http://wiki.ros.org/>
- [2] 8 reasons why you should use ROS for robotics projects
<https://niryo.com/2018/01/8-reasons-use-ros-robotics-projects/>
- [3] 15 reasons to use the robot operating system ros
<https://www.intorobotics.com/15-reasons-to-use-the-robot-operating-system-ros/>
- [4] Anil Mahtani, Luis Sánchez, Enrique Fernández, Aaron Martinez *Effective Robotics Programming with ROS Third Edition*. Birmingham, 2016
- [5] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim *ROS Robot Programming-From the basic concept to practical programming and robot application*. Dec 22, 2017
- [6] Enrique Fernández, Luis Sánchez Crespo, Anil Mahtani, Aaron Martinez *Learning ROS for Robotics Programming Second Edition*. Birmingham, 2015
- [7] Lentin Joseph *Mastering ROS for Robotics Programming*. Birmingham, 2015
- [8] Morgan Quigley, Brian Gerkey, and William D. Smart *Programming Robots with ROS*. O'Reilly, 2010
- [9] R. PATRICK GOEBEL *ROS By Example-A Do-It-Yourself Guide to the Robot Operating System*. 2012
- [10] Jason M. O’Kane *A Gentle Introduction to ROS*. University of South Carolina, 2014

-
- [11] Joao Gomes, Francisco Marques, Andre Lourenco, Ricardo Mendonca, Pedro Santana, Jose Barata *Gaze-Directed Telemetry in High Latency Wireless Communications: The Case of Robot Teleoperation*.
- [12] Giuseppe Conte, David Scaradozzi, Laura Sorbi, Luca Panebianco, Daniele Mannocchi *ROS Multi-agent Structure for Autonomous Surface Vehicles*. Università Politecnica delle Marche
- [13] Abhishek B, Gautham S, Varun Rufus Raj Samuel D, Keshav k, U.P. Vignesh, Shyam R Nair *ROS based Stereo Vision System for Autonomous Vehicle*. International Conference on Power, Control, Signals and Instrumentation Engineering 2017
- [14] SSH COMMAND
<https://www.ssh.com/ssh/command/>
- [15] Secure Shell (SSH)
<https://searchsecurity.techtarget.com/definition/Secure-Shell>
- [16] How To Set Up SSH Keys
<https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2>
- [17] Understanding the SSH Encryption and Connection Process
<https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process>
- [18] Basic SSH Commands
<https://www.hostinger.com/tutorials/ssh/basic-ssh-commands>
- [19] Using SSH key in your web development workflow
<https://studiok40.com/ssh-key-workflow/>
- [20] Database
<https://www.britannica.com/technology/database>
- [21] What is Database? What is SQL?
<https://www.guru99.com/introduction-to-database-sql.html>
- [22] database (DB)
<https://searchsqlserver.techtarget.com/definition/database>

-
- [23] Types of database management system and their evolution
<https://www.analyticsvidhya.com/blog/2014/11/types-databases-evolution/>
- [24] Structured Query Language (SQL)
<https://www.techopedia.com/definition/1245/structured-query-language-sql>
- [25] SQL (Structured Query Language)
<https://www.ntchosting.com/encyclopedia/databases/structured-query-language/>
- [26] What is SQL? Structured Query Language explained
<https://www.infoworld.com/article/3219795/what-is-sql-structured-query-language-explained.html>
- [27] SQL Basics: Working with Databases
<https://www.dataquest.io/blog/sql-basics/>
- [28] What-is-SQL-Server
<https://www.databasejournal.com/features/mssql/article.php/3769211/What-is-SQL-Server.htm>
- [29] How to set up and learn Sql on Mac
<https://www.macworld.co.uk/how-to/mac-software/how-set-up-learn-sql-in-mac-os-x-3638150/>
- [30] What is Computer Vision?
<https://hayo.io/computer-vision/>
- [31] Computer vision
https://en.wikipedia.org/wiki/Computer_vision
- [32] OpenCV
<https://docs.opencv.org/master/index.html>
- [33] Detecting cats in images with OpenCV
<https://www.pyimagesearch.com/2016/06/20/detecting-cats-in-images-with-opencv/>

- [34] Understanding YOLO
<https://hackernoon.com/understanding-yolo-f5a74bbc7967>
- [35] Gary Bradski and Adrian Kaehler *Learning OpenCV*. O'Reilly, September 2008
- [36] Minh Nguyen *MACHINE LEARNING:DEVELOPING AN IMAGE RECOGNITION PROGRAM – with Python, Scikit Learn and OpenCV*. TURKU UNIVERSITY OF APPLIED SCIENCES, 2016
- [37] Paul Viola, Michael Jones *Rapid Object Detection using a Boosted Cascade of Simple Features*. ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001
- [38] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi *You Only Look Once:Unified, Real-Time Object Detection*.
- [39] TRAIN YOUR OWN OPENCV HAAR CLASSIFIER
<https://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>
- [40] Face detection using OpenCV and Python: A beginner's guide
<https://www.superdatascience.com/blogs/opencv-face-detection>
- [41] Cascading classifiers
https://en.wikipedia.org/wiki/Cascading_classifiers
- [42] Gentle guide on how YOLO Object Localization works with Keras (Part 2)
<https://heartbeat.fritz.ai/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2-65fe59ac12d>
- [43] HTML
<https://en.wikipedia.org/wiki/HTML>
- [44] HTML
<https://www.computerhope.com/jargon/h/html.htm>
- [45] What is HTML?
<https://www.yourhtmlsource.com/starthere/whatishtml.html>

- [46] How Does HTML Work?
<http://www.tech-faq.com/how-does-html-work.html>
- [47] freepik
https://www.freepik.com/free-icon/html-file-with-code-symbol_742687.htm
- [48] Sebastian Raschka *Python Machine Learning*. Birmingham, 2015
- [49] Quickstart-A Minimal Application
<http://flask.pocoo.org/docs/0.12/quickstart/rendering-templates>