CAPITOL TECHNOLOGY UNIVERSITY

1927

Department of Astronautical Engineering
Fusion Laboratory

Nanook Software Documentation

Team sponsored by:

CAPES
Ministério da Educação

GOVERNO FEDERAL BRASIL
PAÍS RICO É PAÍS SEM POBREZA

Names:
Ana Beatriz Prudêncio de Almeida Rebouças
Nadson Renan Tomé de Sousa
Thiago Alves Lima

SOFTWARE PART

The analysis of the code consists in two parts: the embedded code (running on the robot CPU) and the computer software (running in the computer with the user's interface software). The computer software was written in C#, while the embedded software was written in C++. Both are object oriented languages. The development environment of the computer software was the Visual Studios 2008. We have used the backup 6 of the computer software on this documentation because we believe that is the latest version.

Computer Software

Software Organization

As the software was written in an objected oriented language, it is organized in objects, classes, and methods. Below, we have the Class Diagram that gives us an insight of how the software is organized. From the Class Diagram, we have gathered some parameters (classes) to explain how the software interact with the robot, such as Form, Class, Struct, and Enum.

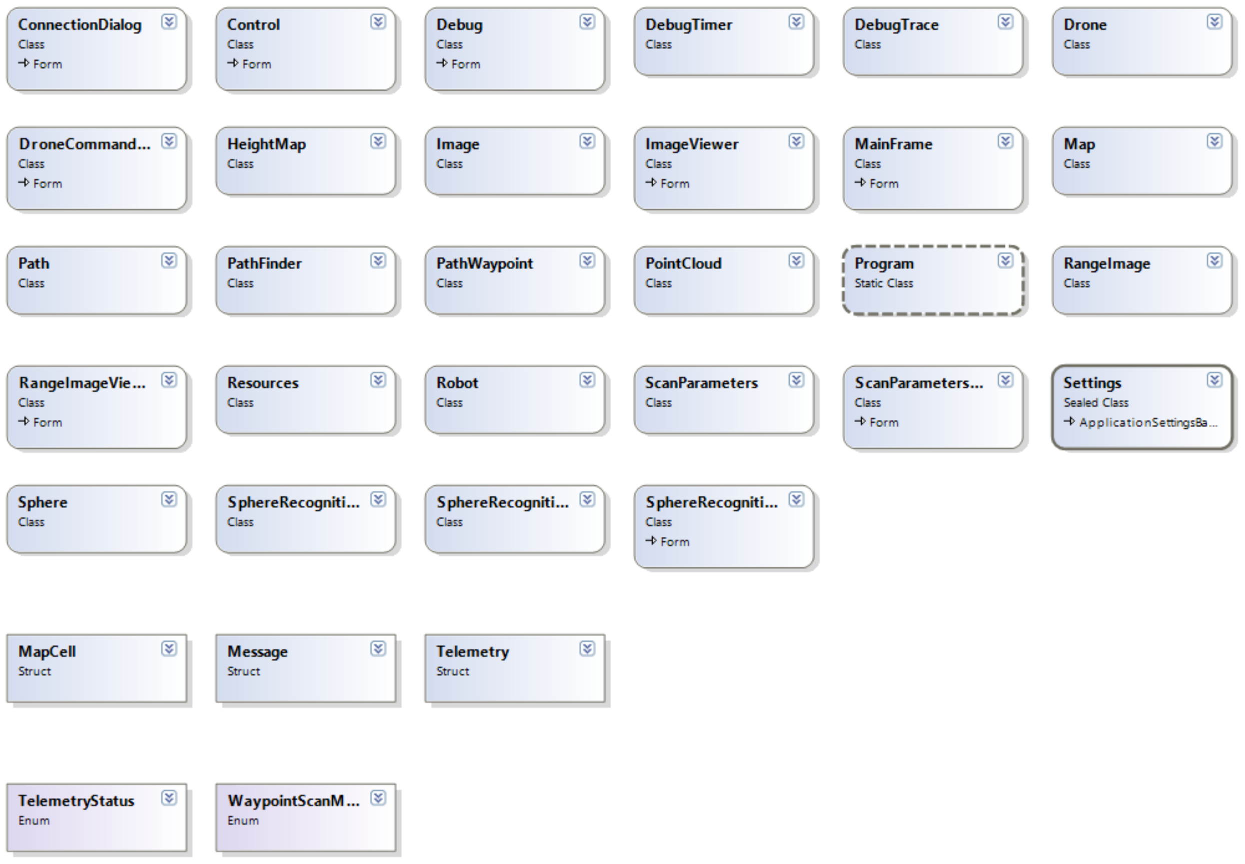
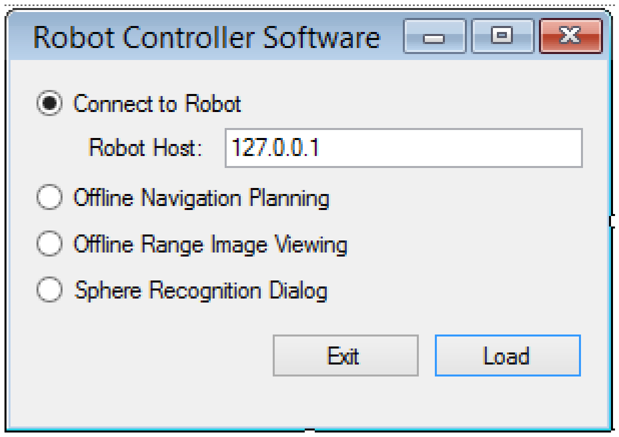


Figure 1. Class Diagram of the Nanook Computer Software

1.Form

The class Form is related to a window or dialog box used in GUI interface for applications. Essentially, we can use buttons, text boxes, tabs, and important graphics features in our software. The ConnectionDialog, Control, Debug, DroneCommandList, MainFrame, RangeImageViewer, ScanParametersViewer, and SphereRecognitionView classes extend from Form. These classes inherit the properties and characteristics from the Form class and add to them their specificities.

1.1. Connection_Dialog Form


Robot Controller Software

(●) Connect to Robot

Robot Host: 127.0.0.1

( ) Offline Navigation Planning

( ) Offline Range Image Viewing

( ) Sphere Recognition Dialog

Figure 2. Clonnection_Dialog Form

This form is the first layer of the user interaction to the robot. When the user first start the computer software, the window above pops up and we have to choose the mode that we are going to use the software through the choice of the selection of radio buttons: Connect to robot, Offline Navigation Planning, Offline Range Image Viewing, and Sphere Recognition Dialog. The choice of checking one of the radio buttons leads to a different user experience, after we hit the button "Load".

1.1.1. Connect to Robot

For default, the first radio button is checked (Connect to Robot) and the IP address is 127.0.0.1, which is the computer's IP. Then we can directly delete the default IP and write the robot IP address to connect the software remotely to the robot. After we click the Load button, the Main, in which is called Robot Controller, and Control forms pop up on the screen.

1.1.2. Offline Navigation Planning

This mode starts with an issue on the code that is related to we cannot really work on this mode because the software is showing mistakes on the code due to the fact that the control form is opened and the variable "m_Robot" was not defined. The code was not implemented to work on the mode because the user starts the Navigation Planning without defining the object. Besides, the user cannot access the previous navigation data because the software does not have a way to save the navigation data and it is not implemented to not receive data from the robot on this mode.

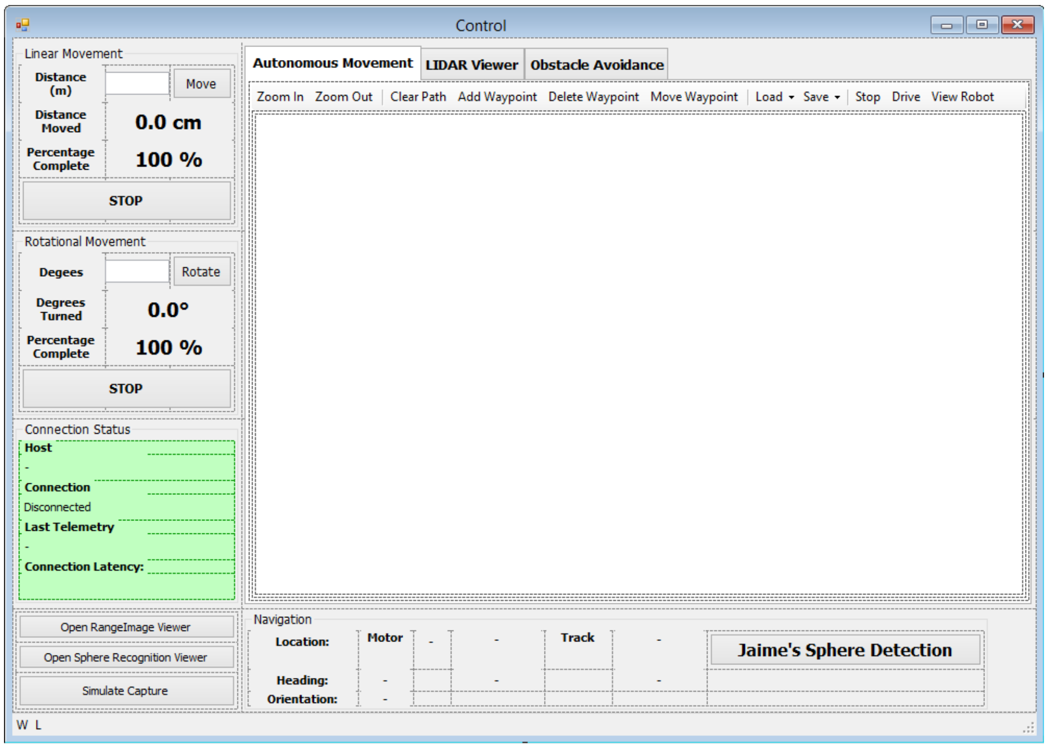
1.1.3. Offline Range Image Viewing

The offline RangeImage Viewing is used to load a ".RangeImage" file which is the picture of the respective Scanning. After loading the file, we can zoom in or out and also we can change among the 4 style of visualisation, point cloud, colored by height, colored by distance and flat image. However, just the *Colored by Distance* and *Colored by Height* are properly working,

1.1.4. Sphere Recognition Dialog

The Recognition Dialog is an offline mode that loads a ".RangeImage" file that was acquired after the scanning inspection. The Sphere Recognition Dialog has an algorithm to find the sphere shapes on the ".RangeImage" file. There are some TextBoxes that helps the user to set the image processing settings and also a button to test the previous scanning from Nanook.

1.2. Control Form



Distance (m)
Distance Moved 0.0 cm
Percentage Complete 100 %
Degees
Degrees Turned 0.0°
Percentage Complete 100 %
Host -
Connection Disconnected
Last Telemetry -
Connection Latency:
Location: Motor - - Track -
Heading: - - -
Orientation: -

Figure 3. Default Control Form.

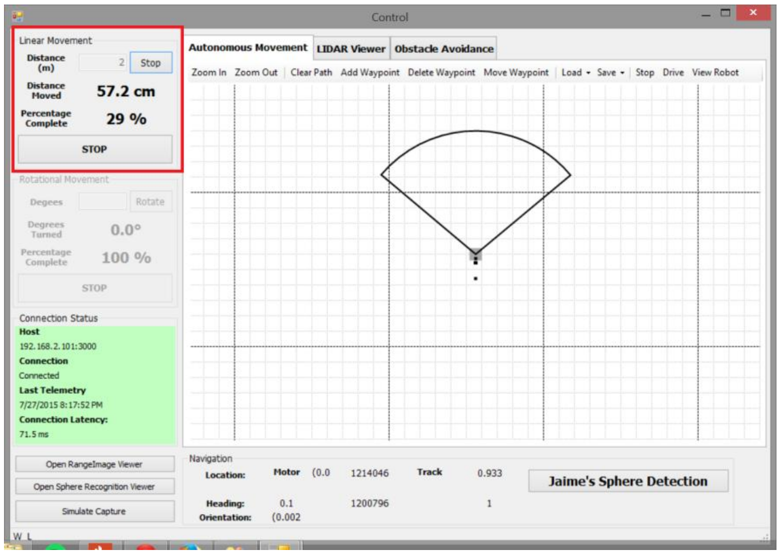
On the Control Form, we can move the robot using the GroupBox called Linear Movement and also the Rotational Movement.

In the beginning, for default, the labels related to the "Distance Moved", "Degree Turned", and the two "Percentage Complete"on the Control Form are set as the table below:

Table 1. Control Form Labels default.

Before Moving or Turning	
Linear Movement	Rotational Movement
Distance Moved = 0.0 cm	Degree Turned = 0.0°
Percentage Complete = 100%	Percentage Complete = 100%
After Moving or Turning	
Linear Movement	Rotational Movement
Distance Moved = 0.0 cm	Degree Turned = 0.0°
Percentage Complete = 0%	Percentage Complete = 0%

On the Linear Movement, the user can write the value of the distance that wants to move the robot on the TextBox beside the button "Move". By clicking on the button "Move", Nanook starts to move and the values of the "Distance Moved" and "Percentage Complete" change on the Control Form as the user can see on the picture below:



Linear Movement
Distance (m): 2
Distance Moved: 57.2 cm
Percentage Complete: 29 %

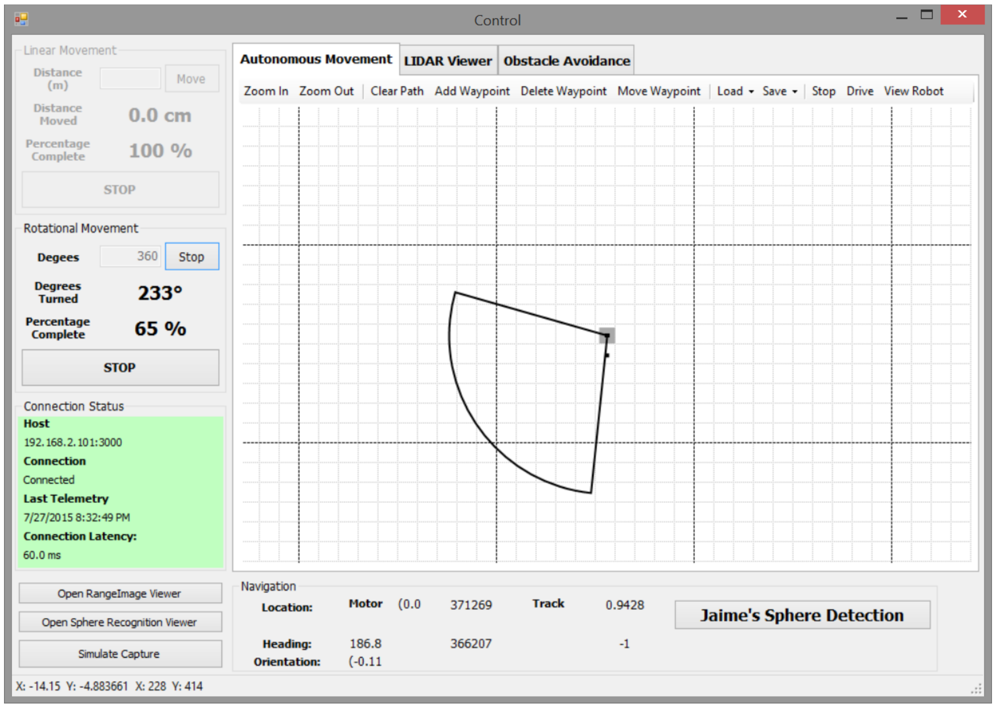
Rotational Movement
Degees:
Degrees Turned: 0.0°
Percentage Complete: 100 %

Connection Status
Host
192.168.2.101:3000
Connection
Connected
Last Telemetry
7/27/2015 8:17:52 PM
Connection Latency:
71.5 ms

Navigation
Location: Motor (0.0 1214046 Track 0.933
Heading: 0.1 1200796 1
Orientation: (0.002

Figure 4. Linear Movement GroupBox alteration on Nanook's motion.

The user can verify the same behavior on the Rotational Movement. For example, the picture below illustrates the moment which Nanook is turning 360° around:



Linear Movement
Distance (m)
Distance Moved 0.0 cm
Percentage Complete 100 %

Rotational Movement
Degees 360
Degrees Turned 233°
Percentage Complete 65 %

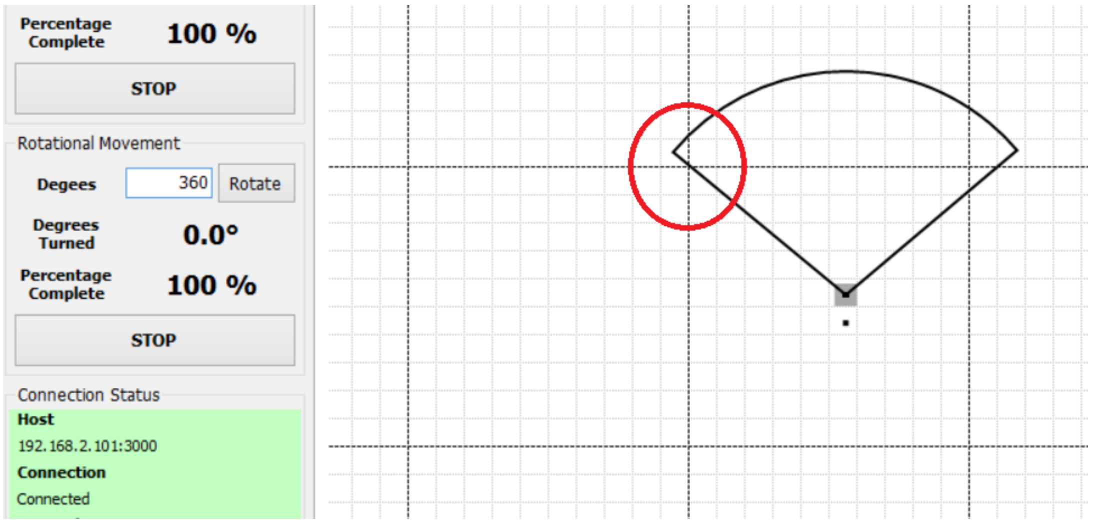
Connection Status
Host
192.168.2.101:3000
Connection
Connected
Last Telemetry
7/27/2015 8:32:49 PM
Connection Latency:
60.0 ms

Navigation
Location: Motor (0.0 371269 Track 0.9428
Heading: 186.8 366207 -1
Orientation: (-0.11

X: -14.15 Y: -4.883661 X: 228 Y: 414

Figure 5. Control Form Screen while Nanook is turning around.

For the linear movement, it is hard to verify if there is any calibration mistake just by checking on the screen. However, for the rotational movement, it is easier to identify an evident error because of the view angle position on the screen. The following 2 pictures show the calibration error:



Percentage Complete 100 %

Rotational Movement
Degees 360
Degrees Turned 0.0°
Percentage Complete 100 %

Connection Status
Host
192.168.2.101:3000
Connection
Connected

Figure 6. Before 360° turn.

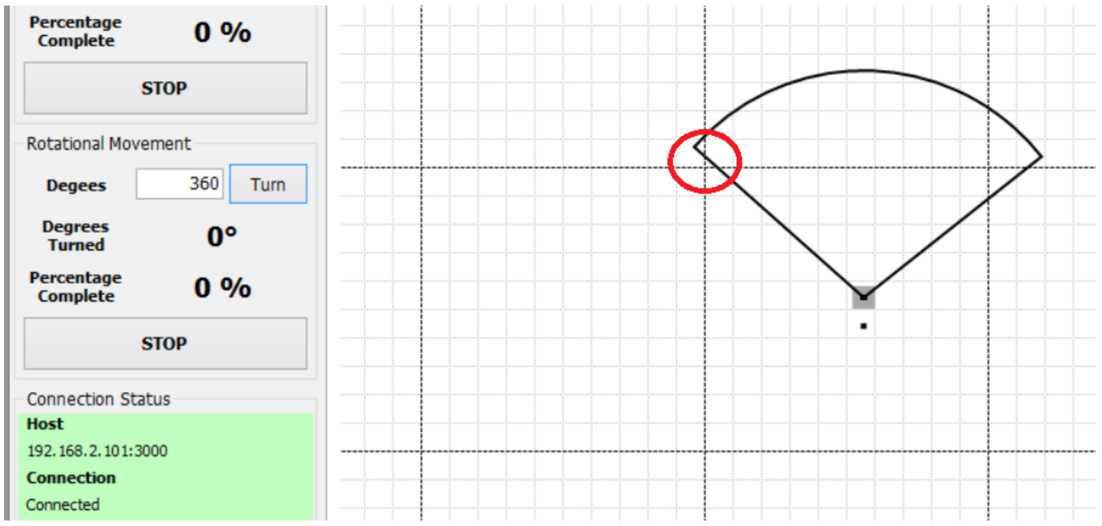


Figure 7. After 360° turn.

1.3. Debug Form

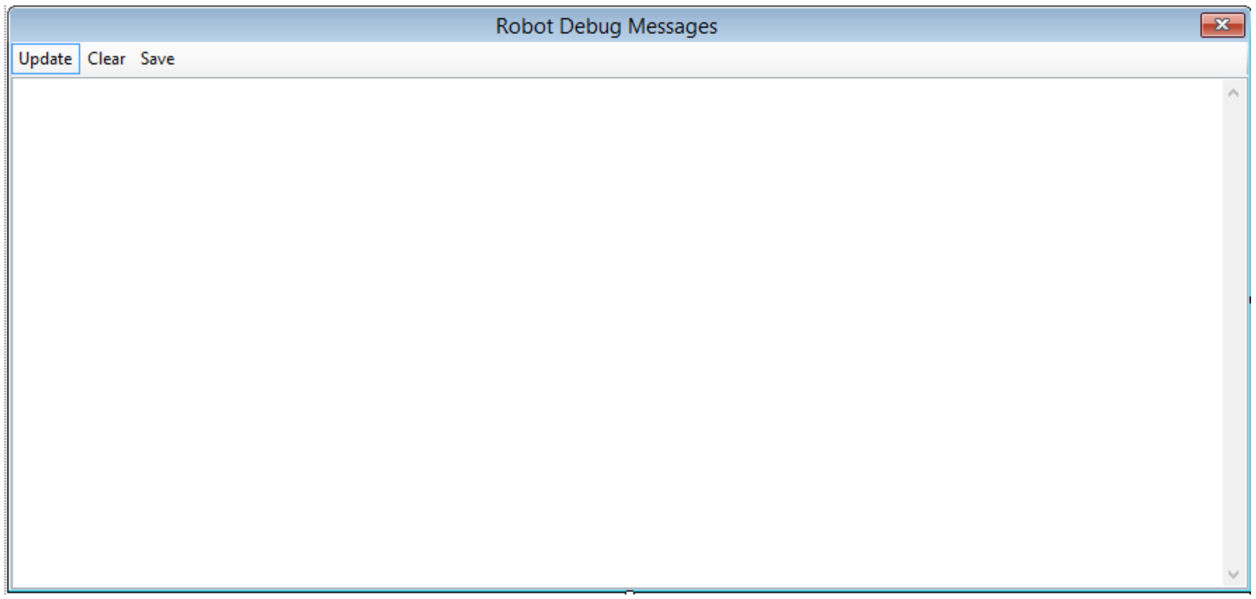


Figure 8. Debug Form

The Debug Form was used by the programmers with the aim of being able to use that interface as a tool to check data received from the embedded software, and then analyzing if the data was the one expected.

1.4. DroneCommandList Form

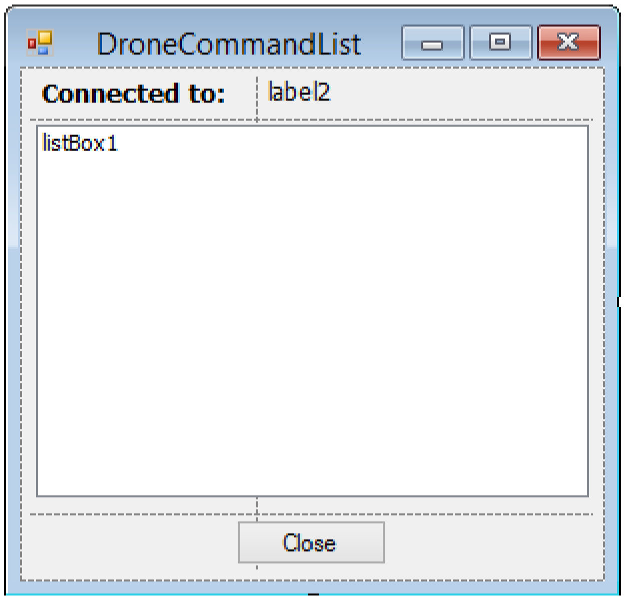


Figure 9. DroneCommandList form

The DroneCommandList is a Form in the computer software that is useless. During Nanook's development time, the programmers have not finished this Form. The purpose of this part of the software has not been specified on the code, but it supposes this Forms had been designed or at least thought to allow the communication between Nanook with Penguin #1 and #2, the workerbots.

1.5. ImageViewer Form

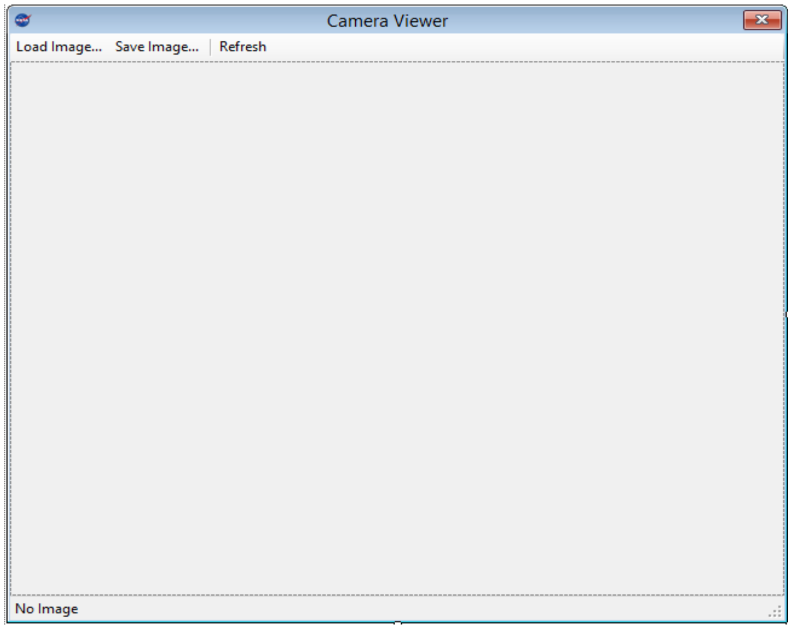
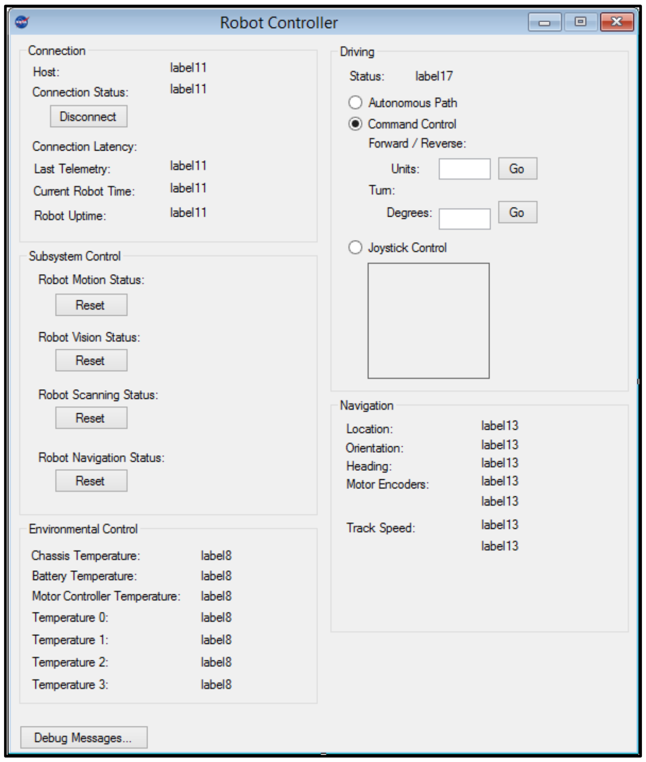


Figure 10. Image Viwer Form.

The ImageViewer is actually not being used. It was probably designed to help in a possible obstacle avoidance application, which ended up not being developed. As the ImageViewer Form was named as “Camera Viewer”, it was probably supposed to show a real-timing images from a camera on the screen.

1.6. Main Form



Connection

Host:	label11
Connection Status:	label11
Connection Latency:	
Last Telemetry:	label11
Current Robot Time:	label11
Robot Uptime:	label11

Subsystem Control

Robot Motion Status:

Robot Vision Status:

Robot Scanning Status:

Robot Navigation Status:

Environmental Control

Chassis Temperature:	label8
Battery Temperature:	label8
Motor Controller Temperature:	label8
Temperature 0:	label8
Temperature 1:	label8
Temperature 2:	label8
Temperature 3:	label8

Driving

Status: label17

Autonomous Path [✓ Command Control] Joystick Control

Forward / Reverse:

Units:

Turn:

Degrees:

Navigation

Location:	label13
Orientation:	label13
Heading:	label13
Motor Encoders:	label13
	label13
Track Speed:	label13
	label13

Figure 11. Main Form

The Main Form is used to observe the effects from the environment on the Nanook (temperature), to check the Wi-Fi connection and its quality, and also to check and control the linear or rotational movement. In this Form, there are 6 GroupBoxes. One of them is called connection, the other ones are connection latency, subsystem control, environmental control, driving and navigation.

1.6.1. Connection

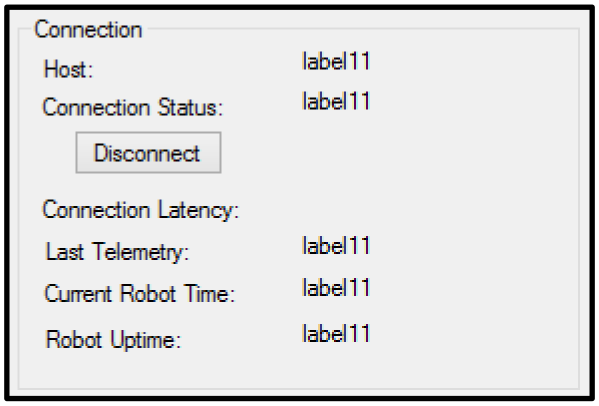


Figure 12. Connection GroupBox

“Connection” is responsible for Connect or Disconnect the software to the robot, informing to the user about the connection status, the host IP address, and the latency as well.

Connection Latency is responsible to inform to the user the connection quality, giving the data response time between Nanook and the computer software.

1.6.2. Subsystem control

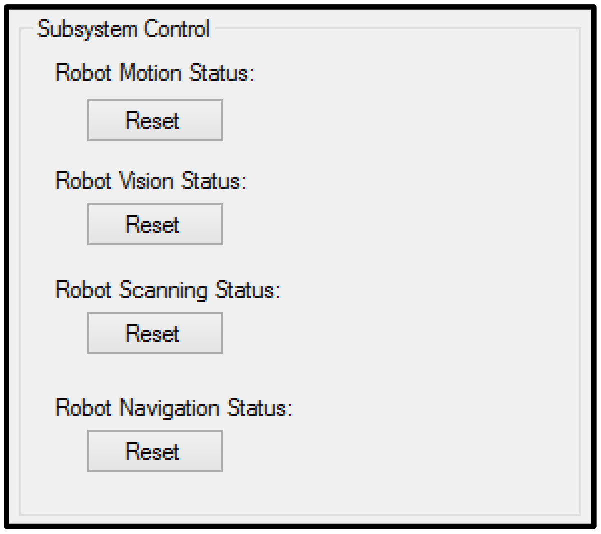
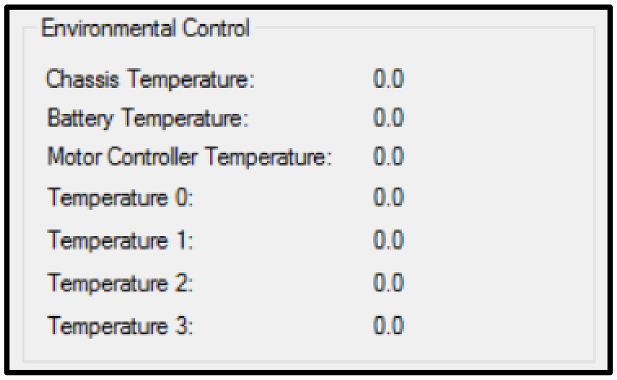


Figure 13. Subsystem Control GroupBox

“Subsystem Control” is the GroupBox that contains the reset buttons for robot motion, vision, scanning, and navigation. Clicking on those buttons does not leads to any event because the previous programmers have not created any instructions for this kind of action.

The buttons in the GroupBox were not implemented. During the computer software inspection, it was possible to realize that the buttons were maintained the way as were created, changing just the name for “Reset”.

1.6.3. Environmental Control


Environmental Control

Chassis Temperature:	0.0
Battery Temperature:	0.0
Motor Controller Temperature:	0.0
Temperature 0:	0.0
Temperature 1:	0.0
Temperature 2:	0.0
Temperature 3:	0.0

Figure 14. Environmental Control GroupBox

The Nanook's references of temperature have not been implemented. The Environmental Control GroupBox has seven different types of temperatures that have not been measured updated, first because there is no code to process this information from Nanook, second because there is no hardware for that purpose. The software was supposed to gather data from Nanook about temperature by from some Wi-Fi socket, but it is actually not doing it. As professor Patrick Stakem told us, this could be implemented by using arduino. Arduino sends the information to the embedded CPU and then send to the computer through Wi-Fi. The next group that will work on this project could implement this feature.

1.6.4. Driving

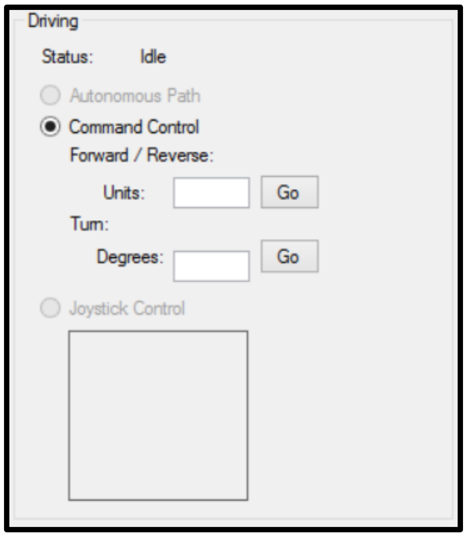
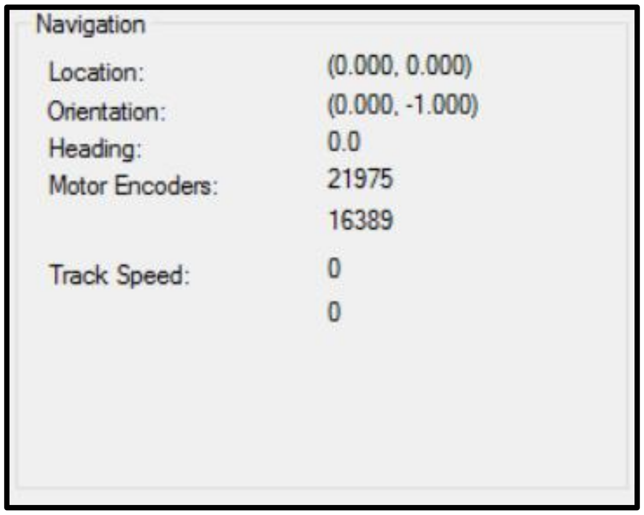

Driving
Status: Idle
Autonomous Path
Command Control
Forward / Reverse:
Units: Go
Turn:
Degrees: Go
Joystick Control

Figure 15. Driving GroupBox

The Driving GroupBox works similarly to both ***Linear*** and ***Rotational*** **Movement** on Control Form.

There are some RadioButtons that select the type of driving. However just the Command Control is implemented. The Autonomous Path has not been developed and the Joystick Control is unable because probably it has been designed to work with the ImageViewer that could give the real-timing images from the robot.

1.6.5. Navigation



Location:	(0.000, 0.000)
Orientation:	(0.000, -1.000)
Heading:	0.0
Motor Encoders:	21975
	16389
Track Speed:	0
	0

Figure 16. Navigation GroupBox

The Navigation GroupBox is relevant to inform to the user about the navigation data, for instance, the robot coordinates, orientation,   to both ***Linear*** and ***Rotational*** **Movement** on Control Form.

There are some RadioButtons that select the type of driving. However just the Command Control is implemented. The Autonomous Path has not been developed and the Joystick Control is unable because probably it has been designed to work with the ImageViewer that could give the real-timing images from the robot.

1.7. RangeImageViewer Form

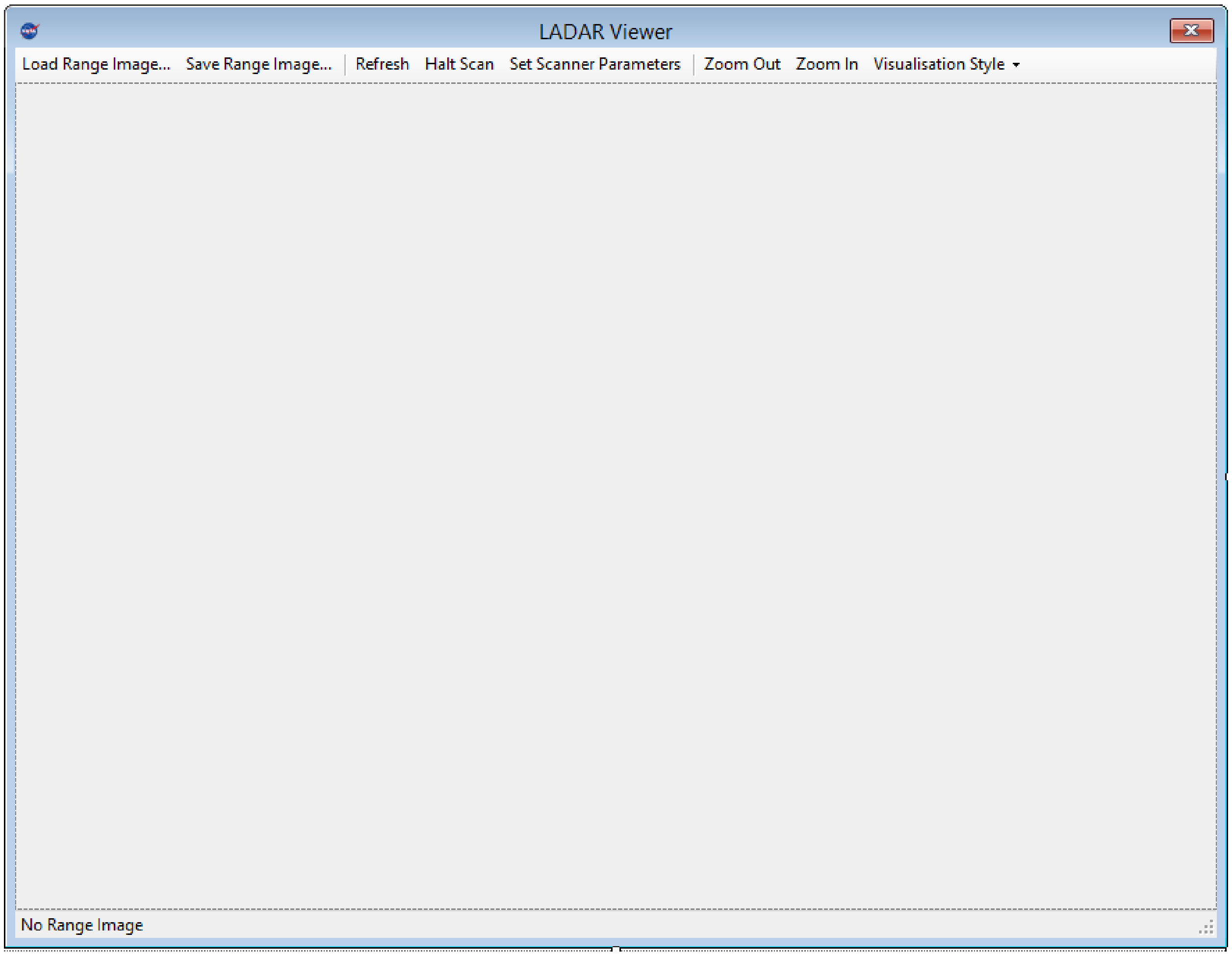


Figure 17. Class Diagram of the Nanook Computer Software

RangeImageViewer is the interface responsible for showing to the users the gathered picture from the LIDAR. There are 8 buttons on this window called Load Range Image, Save Range Image, Refresh, Halt Scan, Set Scanner Parameters, Zoom Out, Zoom In, Visualisation Style.

The *Load Range Image* button allows the user to open a previous saved image.

The *Save Range Image* button allows the user to save the picture obtained on the scanning.

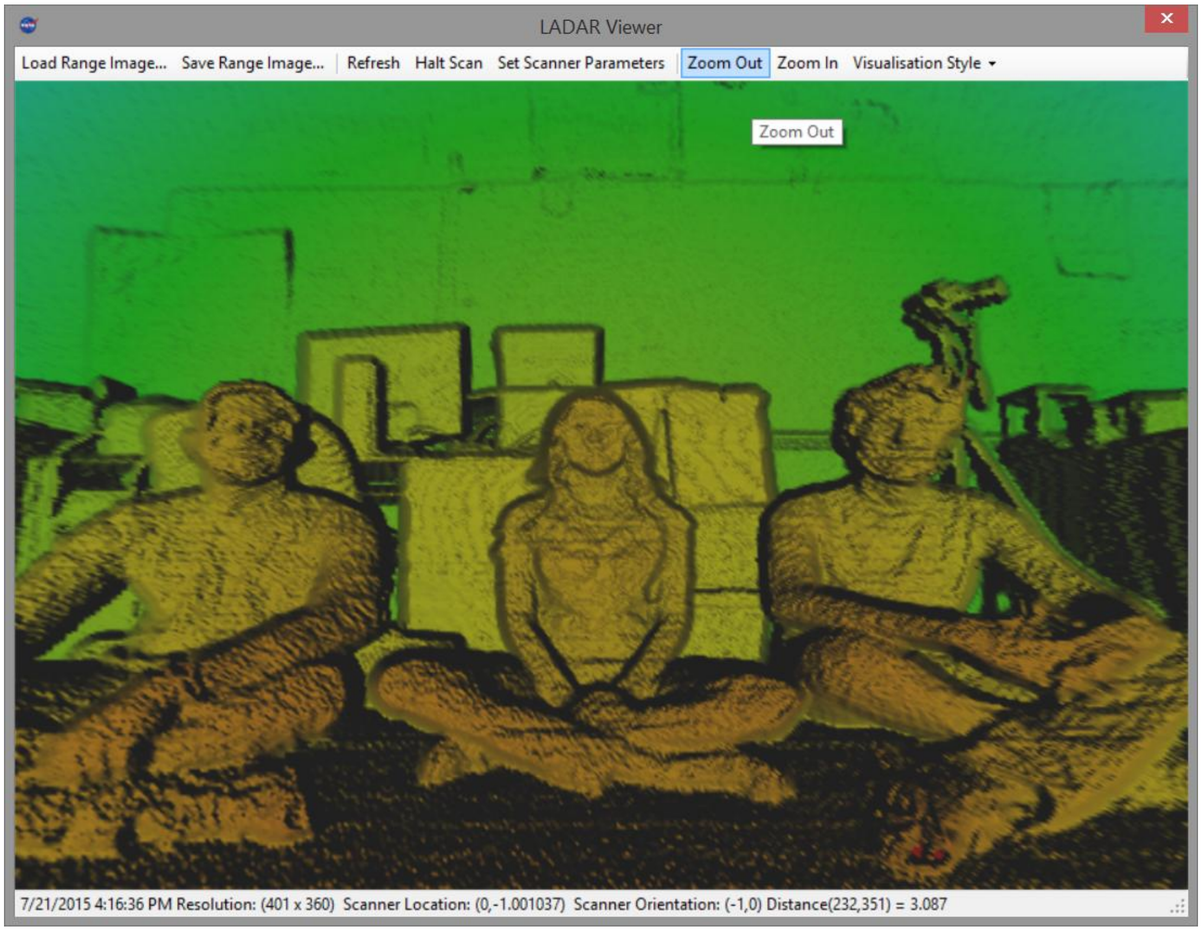
The *Refresh* button executes another scanning process.

The *Halt Scan* button stops the LIDAR scanning activity.

The *Set Scanner Parameters* button opens the Form called ScanParametersViewer where the user can choose the scan parameters.

The *Zoom Out* and *Zoom In* buttons make the user's interaction to the obtained picture, approximating or getting far from the picture.

By clicking on the button *Refresh*, the user is able to obtain the picture below from the LIDAR:


LADAR Viewer
Load Range Image... Save Range Image... Refresh Halt Scan Set Scanner Parameters Zoom Out Zoom In Visualisation Style
Zoom Out

7/21/2015 4:16:36 PM Resolution: (401 x 360) Scanner Location: (0,-1.001037) Scanner Orientation: (-1,0) Distance(232,351) = 3.087

Figure 18. Colored by High picture from the LIDAR.

1.8. ScanParametersViewer Form

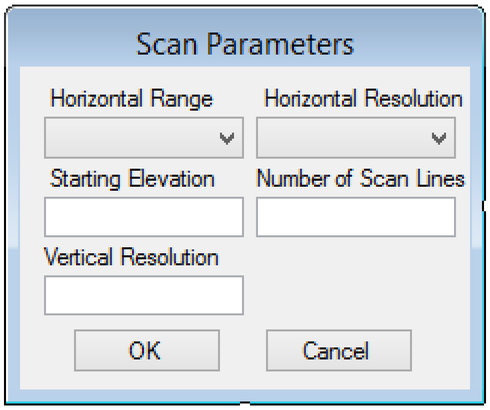


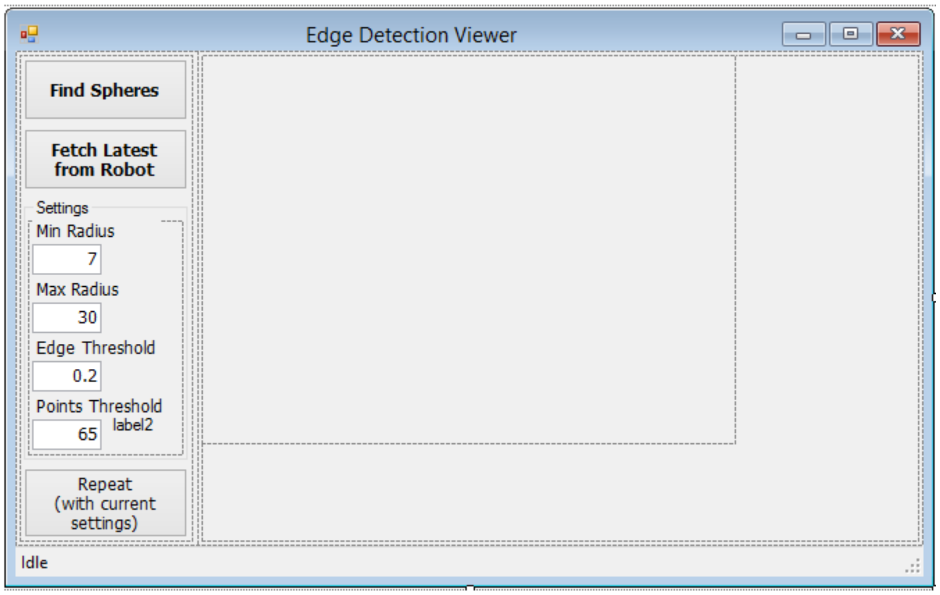
Figure 19. ScanParametersViewer Form

ScanParametersViewer is the interface responsible for set the LIDAR scan parameters. The scan parameters are important for the user select the desired resolution. The parameter are called Horizontal Range, Horizontal Resolution, Starting Elevation, Number of Scan Lines, and Vertical Resolution.

Table 2. Default Parameters

Default Configuration			
Parameters	Value	Parameters	Value
Horizontal Range	100	Number of Scan Lines	360
Horizontal Resolution	0.25	Vertical Resolution	0.25
Starting Elevation	-45°	----	----

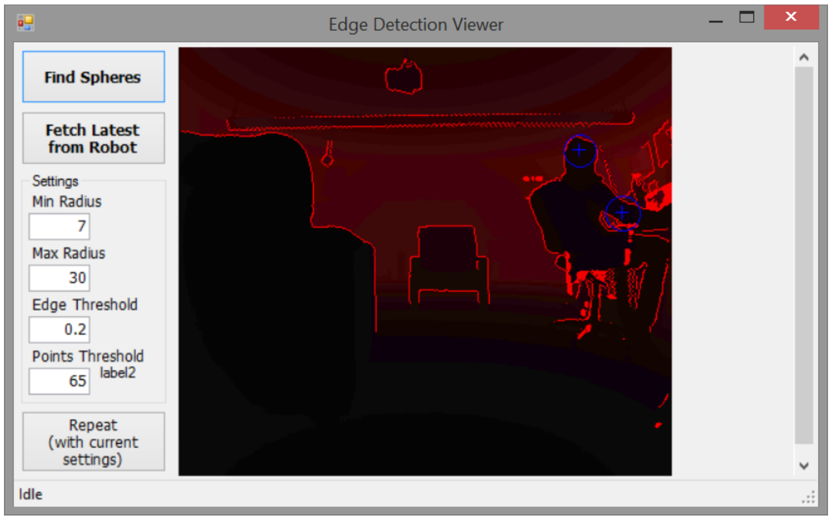
1.9. SphereRecognitionView Form



Min Radius: 7
Max Radius: 30
Edge Threshold: 0.2
Points Threshold: 65

Figure 20. Sphere Recognition interface.

The SphereRecognitionView is the interface responsible for the identification of Nanook's workerbots. On this interface, the software can show to the user the sphere by circulating the location on the picture where the sphere is. For example, the picture below shows what was supposed to be a sphere circulated by a blue circle:



Settings
Min Radius 7
Max Radius 30
Edge Threshold 0.2
Points Threshold 65
label2

Figure 21. Sphere detection by circulating.

2. CLASSES

A class is a code component that allows someone to create his/her own custom types in order to group variables, methods, and events that characterize the cluster. Classes, therefore, define data and behavior.

As stated in [x3], classes are one of the most important components in an object oriented language: “A program is an abstract machine. As it executes, its parts are in movement, in constant motion, moving towards a result. Its parts are called classes.”

In the Nanook’s computer software we have identified the following classes: ConnectionDialog, Control, Debug, DebugTimer, DebugTrace, Drone, DroneCommandList, HeightMap, Image, ImageViwer, MainFrame, Map, Naviagtion, NavigationBAK, Path, PathWaypoint, PointCloud, Program, RangeImage, RangeImageViewer, Robot, ScanParameters, ScanParametersViewer, Sphere, SphereRecognition, SphereRecognition2, SphereRecognitionView.

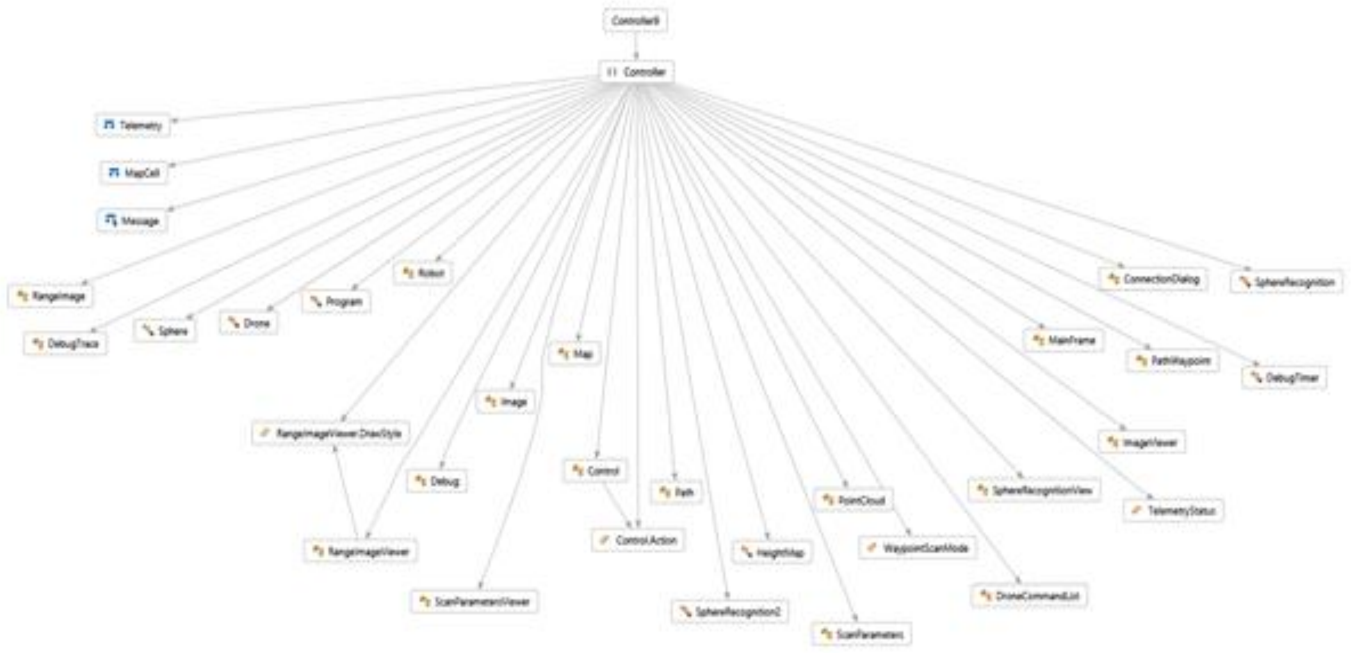


Figure 22. UML Diagram for Classes of the Nanook Computer Software

2.2. Partial Class

Partial Classes are a methodology chosen by programmers of large projects to physically split the definition of certain class in multiple source files.[x5] Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled. [x6]

Windows Forms are one example of Partial Class because code can be added to them without having to recreate the source file. The user can create code that uses these classes without having to modify the file created by Visual Studio.

As Nanook was a project done by several people, partial classes were extensively used. Below, some of the classes defined using partial classes are described.

2.2.1. ConnectionDialog

ConnectionDialog is declared in the ConnectionDialog.cs and ConnectionDialog.Designer.cs, as shown in the table below.

Table 3. ConnectionDialog Class

Class	Sintax	File
ConnectionDialog	public partial class ConnectionDialog : Form { … }	ConnectionDialog.cs
	partial class ConnectionDialog { … }	ConnectionDialog.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:

Inheritance Hierarchy

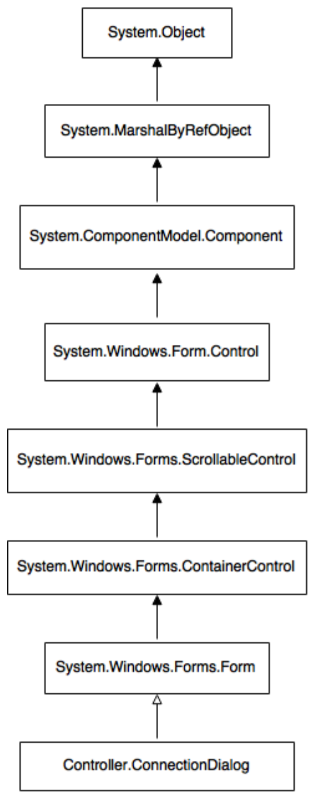

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.ConnectionDialog

Figure 23. ConnectionDialog Hierarchy

By extended, we mean that the class inherits the characteristics (objects and methods) from its mother class, but it add specificities in its definition, becoming a different class.

The ConnectionDialog class is related to the connection options. There are four modes availables to connect: connecting to the robot by providing its IP, offline navigation planning, offline range image viewing, sphere recognition dialog.

Using event handler, this class defines the action that will be taken after the user select one of the radio options. Also, the class defines what will happen if the Cancel or Load buttons are clicked.

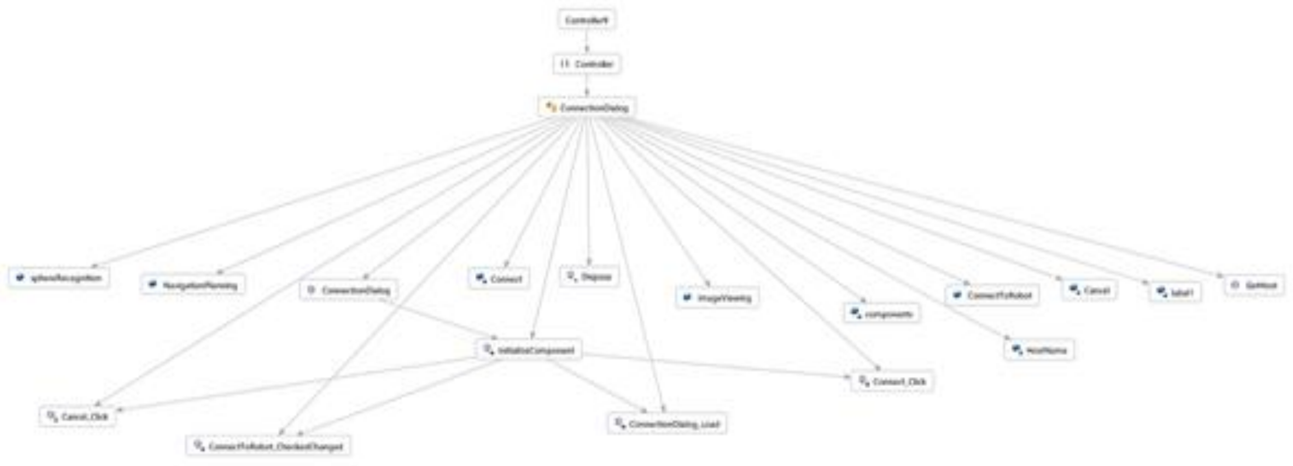


Figure 24. UML diagram for ConnectionDialog.

2.2.2. Control

The Control Form is also a Partial Class.

Control is declared in the Control.cs and Control.Designer.cs, as shown in the table below.

Table 4. Control Class.

Class	Sintax	File

Control	public partial class Control : Form
	{
	…
	}
		Control.cs

	partial class Control
	{
	…
	}
		Control.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:

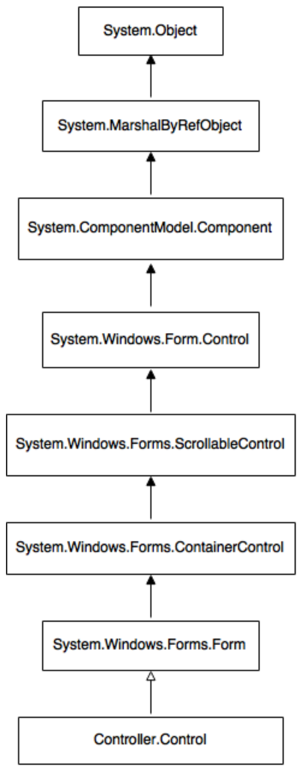

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.Control

Figure 25. Control Hierarchy

The Control class is related to the robot navigation. In this class, the user is able to control the robot and visualize the position of the robot in a room, for example. There are methods related to autonomous navigation, LIDAR scanning request and processing, and navigation control. This is one of the most important classes in the project because the core algorithms were written on this class.

Using event handler, this class defines the action that will be taken after the user click on one of the buttons on this window.

2.2.3. Debug

Debug is declared in the Debug.cs and Debug.Designer.cs, as shown in the table below.

Table 5. Debug Class.

Class	Sintax	File
Debug	public partial class Debug : Form { … }	Debug.cs
	partial class Debug { … }	Debug.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:

Inheritance Hierarchy:

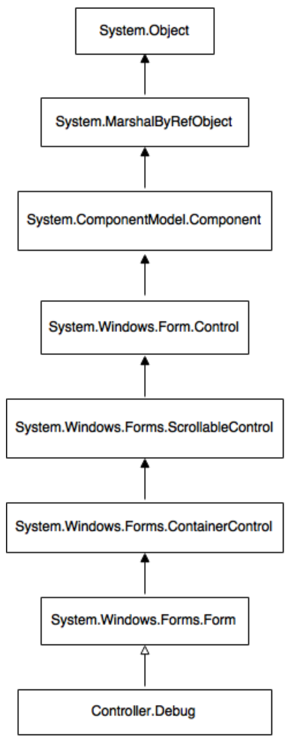

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.Debug

Figure 26. Debug Hierarchy

The Debug class was used to help the programmers to get the messages from the embedded software and show them on the screen when necessary.

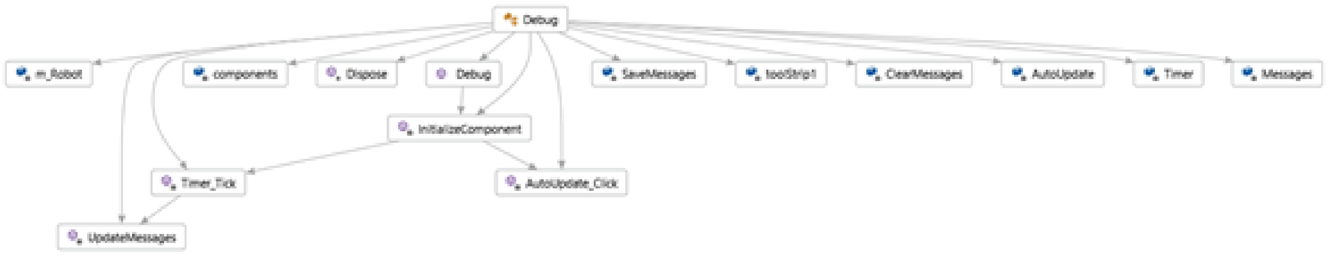


Figure 27. UML diagram for Debug Class.

2.2.4. DroneCommandList

DroneCommandList is declared in the DroneCommandList.cs and DroneCommandList.Designer.cs, as shown in the table below.

Table 6. Control Class.

Class	Sintax	File

Control	public partial class Control : Form
	{
	…
	}
		Control.cs

	partial class Control
	{
	…
	}
		Control.Designer.cs

The DroneCommandList Form follow the same pattern of the previous partial classes. However, the purpose of the class was to show on the screen the other drones IP and its respective commands.

This class is extended from Form, following the complete inheritance hierarchy described below:
Inheritance Hierarchy:

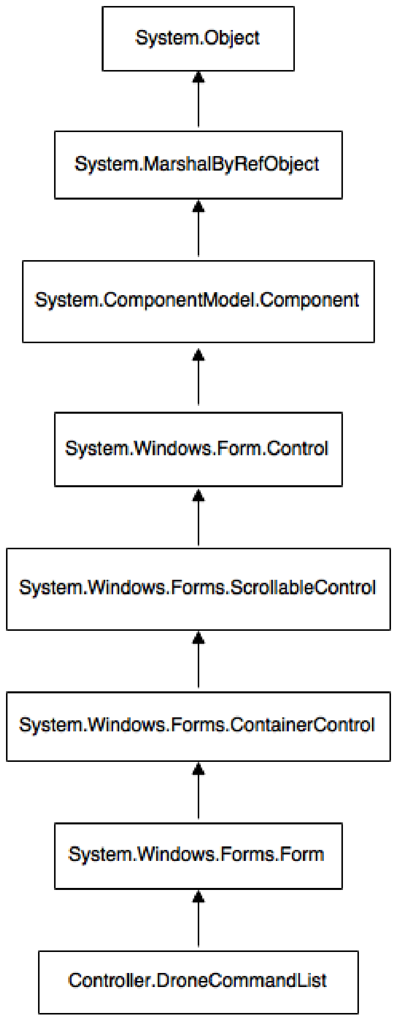

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.DroneCommandList

Figure 28. DroneCommandList Hierarchy

2.2.5. ImageViewer

Debug is declared in the Debug.cs and Debug.Designer.cs, as shown in the table below.

Table 7. ImageViewer Class.

Class	Sintax	File

Control	public partial class Control : Form
	{
	…
	}
		Control.cs

	partial class Control
	{
	…
	}
		Control.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:
Inheritance Hierarchy:

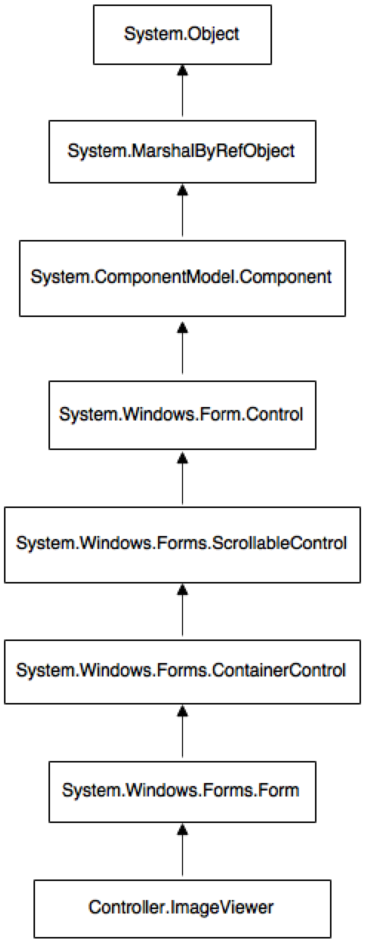

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.ImageViewer

Figure 28. ImageViewer Hierarchy

The ImageViewer partial class was not totally implemented, but it was supposed to be a previous version of the RangeImage or even the implementation of an optical camera to help on the algorithms for the obstacle avoidance which was probably disconsidered throughout the project.

2.2.6. MainFrame

MainFrame is declared in the Main.cs and Main.Designer.cs, as shown in the table below.

Table 8. MainFrame Class.

Class	Sintax	File

MainFrame	public partial class MainFrame : Form
	{
	…
	}
		Main.cs

	partial class MainFrame
	{
	…
	}
		Main.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:
Inheritance Hierarchy:

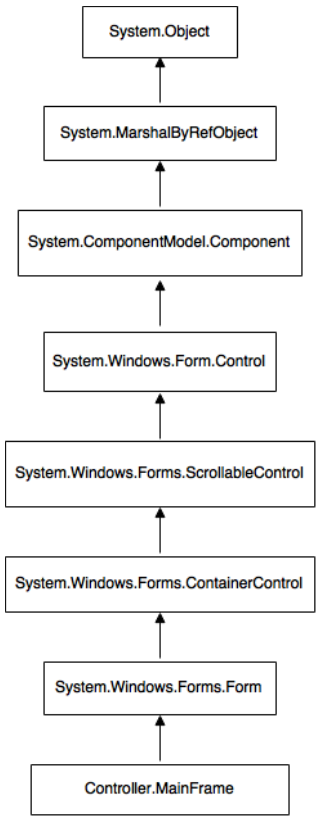

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.MainFrame

Figure 29. MainFrame Hierarchy

The MainFrame Form is a partial class that has some good information about the robot and also some space for control, for instance, environmental temperature. robot speed, status connection and driving by joystick, autonomous, or moving/turning command motion.

2.2.7. RangeImageViewer

Debug is declared in the RangeImageViewer.cs and RangeImageViewer.Designer.cs, as shown in the table below.

Table 9. RangeImageViewer Class.

Class	Sintax	File

RangeImageViewer	`public partial class RangeImageViewer : Form`
	`{`
	`…`
	`}`
		RangeImageViewer.cs

	`partial class RangeImageViewer`
	`{`
	`…`
	`}`
		RangeImageViewer.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:
Inheritance Hierarchy:

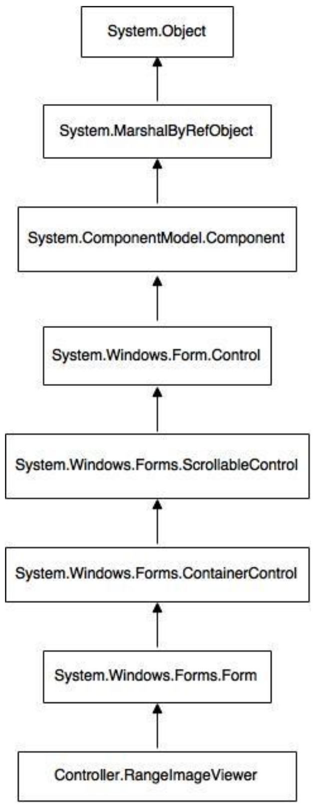

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.RangeImageViewer

Figure 30. RangeImageViewer Hierarchy

The RangeImageViewer is partial class responsible for building and displaying the 3D plotting image from the gathered LIDAR data using the Canvas class assistance.

2.2.8. ScanParametersViewer

ScanParameterViewer is declared in the ScanParameterViewer.cs and ScanParameterViewer.Designer.cs, as shown in the table below.

Table 10. Control Class.

Class	Sintax	File

ScanParameterViewer	public partial class ScanParametersViewer : Form
	{
	…
	}
		ScanParameterViewer.cs

	partial class ScanParametersViewer
	{
	…
	}
		ScanParameterViewer.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:

Inheritance Hierarchy:

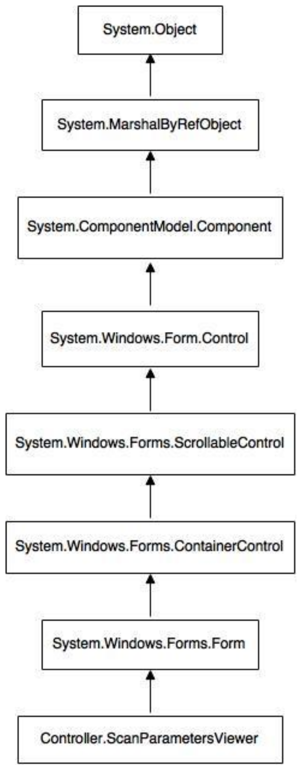

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.ScanParametersViewer

Figure 31. ScanParametersViewer Hierarchy

The ScanParametersViewer is a partial class which helps the user to set the scanning parameters.

2.2.9. SphereRecognitionView

SphereRecognitionView is declared in the SphereRecognitionView.cs and SphereRecognitionView.Designer.cs, as shown in the table below.

Table 11. Control Class.

Class	Sintax	File

SphereRecognitionView	public partial class SphereRecognitionView : Form
	{
	…
	}
		SphereRecognitionView.cs

	partial class SphereRecognitionView
	{
	…
	}
		SphereRecognitionView.Designer.cs

This class is extended from Form, following the complete inheritance hierarchy described below:

Inheritance Hierarchy:

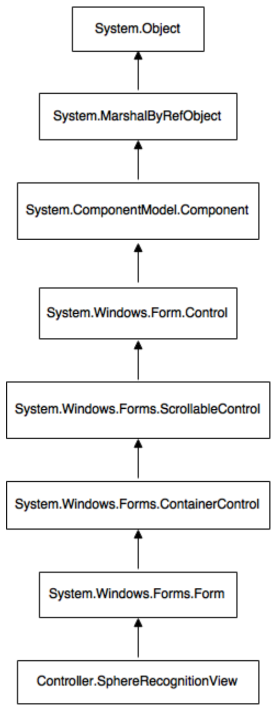

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Form.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form
Controller.SphereRecognitionView

Figure 32. SphereRecognitionView Hierarchy

The SphereRecognitionView Form is a partial class that is responsible for finding the objects with sphere shape. On this partial class and the other ones called, SphereRecognition and SphereRecognition2 there are code with the aim of finding the identifying the sphere of the workerbots, Penguin#1 and Penguin#2.

2.3. Static Class

The Controller has a static class: Program. It is used as *a unit of organization for methods not associated with particular objects* [x9]. Using this class, creating objects to use the related methods is not necessary because we can use methods directly with the class name. Therefore, static classes are not instantiated, which make them useful to just operate on input parameters and do not have to get or set any internal instance fields. [x8]

2.3.1. Program

The program is related to the definition of what happens when the program starts. The initial connection dialog box appears, so the Program redirects to the corresponding action after this start.

2.4. Sealed Class

The sealed keyword in a class means that the class cannot be used as a base class. “Sealed” prevents other classes to be inherited (derived) from the class with this modifier. For example, the sealed class called Settings in the computer software.

3. Structs.

Structs are convenient to cluster small groups of related variables because they are a group of custom value types that can store their respective values.Someone can use a struct instead of creating an entire class for small applications. As described in [x2], “a struct stores its data in its type. It is not allocated separately on the managed heap. Structs often reside on the evaluation stack. Every program uses simple structs. All value types (int, bool, char) are structs.”

The Controller has three structs: MapCell, Message, and Telemetry.

3.1. MapCell

The MapCell structure is defined in Map.cs (declared as public struct MapCell) and the members exposed by the MapCell type are Constructors and Fields, as summarized in the following tables.

3.2. Message

The Message structure is declared by:

```
struct Message
```

and its members are listed in the following table:

Fields:

Table 12.Message Members.

Name
MessageId
Length

3.3. Telemetry

The telemetry structure is declared by:

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Telemetry
```

and its members are listed in the following table .

Table 13.Telemetry members.

Name
ClimateStatus
CurrentTime
Heading
HeaterState0
HeaterState1
HeaterState2
HeaterState3
HeaterState4
HeaterState5
HeaterState6

HeaterState7
ImageDataLength
MapDataLength
MotionStatus
NavigationStatus
PathDataLength
Position
PrimaryAxis
ScanDataLength
StartTime
TargetTemperature0
TargetTemperature1
TargetTemperature2
TargetTemperature3
TargetTemperature4
TargetTemperature5
TargetTemperature6
TargetTemperature7
Temperature0
Temperature1

Temperature2
Temperature3
Temperature4
Temperature5
Temperature6
Temperature7
TrackSpeed0
TrackSpeed1
TrackTicks0
TrackTicks1
VisionStatus

4. Enum

The *Enum* keyword declares an enumeration[x10], which is a list of a set of named constants. The Controller has two enumerations: TelemetryStatus and WaypointScanMode.

TelemetryStatus has the following members: Driving, Idle, MapUpdated, Navigating, PathUpdated, Scanning.

WaypointScanMode has the Full and None members.

Table 14. TelemetryStatus' members.

Name
Driving
Idle
MapUpdated
Navigating

PathUpdated
Scanning

Table 15. WaypointScanMode’s members.

Name
Full
None

SOFTWARE DOCUMENTATION

Complementing the comprehension of the code, a software documentation was developed. Great part of the code was covered and documented in HTML files (using XML comments), as showed in the table below:

Table xx - Compiled Members [x1]

Type	Compiled Members	
	With XML Comment	Total
Namespaces	0	1
Classes	7	18
Structures	2	2
Enumerations	3	4
Methods	76	88
Properties	41	44
Variables (fields)	10	97
Constants	0	16

The only namespace in our software is Controller. Controller has, like already described, 18 classes, 2 structures, and 3 enumerations.

References

[x1] VSdoc

[x2] (http://www.dotnetperls.com/struct)

[x3] (http://www.dotnetperls.com/class)

[x4]https://msdn.microsoft.com/en-us/library/x9afc042.aspx

[x5] http://www.dotnetperls.com/partial

[x6] https://msdn.microsoft.com/en-us/library/wa80x488.aspx

[x7] http://www.dotnetperls.com/class

[x8] https://msdn.microsoft.com/en-us/library/79b3xss3.aspx

[x9] http://stackoverflow.com/questions/241339/when-to-use-static-classes-in-c-sharp

[x10] https://msdn.microsoft.com/en-us/library/sbbt4032.aspx

Embedded System Software

Introduction

The embedded system's software is written in C++, which is also an object oriented language. The software is composed of headers files (.hpp and .h), source files (.cpp), project files (.vcproj) and solution files. The motherboard's software that manages the system is Windows XP Professional.

Headers and Source Files

In C++, header files (.hpp and .h) are not compiled in the project. However, they are very important to connect source files (.cpp), which are the ones that are actually compiled. Firstly, source files are compiled independently, and in a second step they are connected by the compiler to create a project. When a source file wants to important something from another source file, it has to import its equivalent header file instead.

Project Files and Solution Files

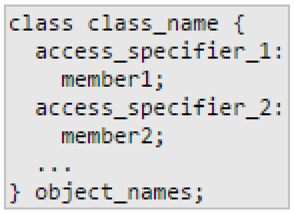
In a higher level of organization, the software is composed of project files (.vcproj) and one solution files (.snl). A Visual C++ project file contains information that is required to build a Visual C++ project. A solution is a grouping of one or more projects that work together to create an application [1].

If you think of it like a tree [2]:

```
.sln
        .vcproj
                .hpp
                .h
                .cpp
```

Classes

Classes in c++ are defined using either the keyword *class* or *struct.* A class is composed of members (data or function declarations) and optionally access specifiers. These specifiers define the access rights for members that follow the members. Specifiers can be private, public or protected. The figure below shows the structure of a class definition.

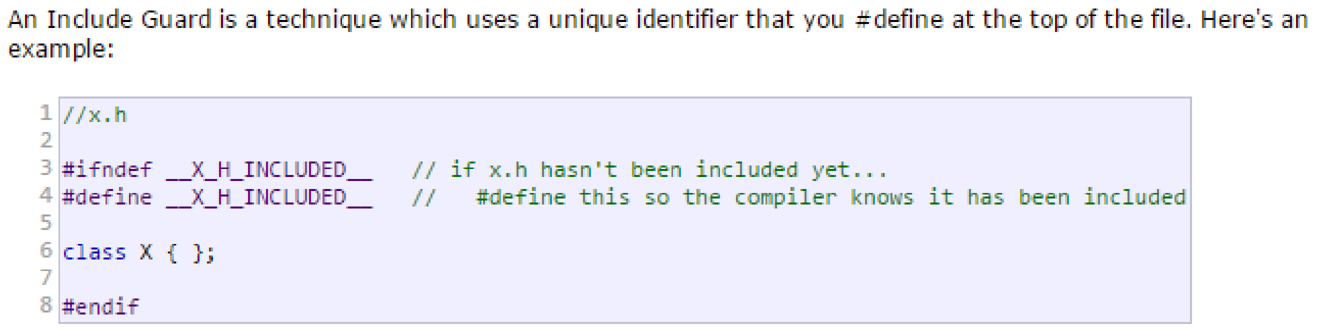


```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} object_names;
```

Figure 33. Class Definition in C++.

Includes and Include Guards

In order to import files in a program, the directive #include is used. Specially in big projects as nanook's embedded software, many header files could get included more than once. This is a problem because it can slow down compilation process by many times and also generate errors. That is why using include guards is a very important practice in C++. For example, suppose that a file x.hpp is included in two other headers, j.hpp and k.hpp. Furthermore, imagine that another file l.hpp includes j.hpp and k.pp. That could generate a problem, because internally x.hpp is also being imported twice in l.hpp. To avoid that, include guards are used, as the example below.



An Include Guard is a technique which uses a unique identifier that you #define at the top of the file. Here's an example:

```
//x.h

#ifndef __X_H_INCLUDED__   // if x.h hasn't been included yet...
#define __X_H_INCLUDED__   //   #define this so the compiler knows it has been included

class X { };

#endif
```

figure 34 - Include guard technique http://www.cplusplus.com/forum/articles/10627

Another way to avoid this would be using *#pragma once.* For example:

x.hpp file

```
#pragma once
class myclass {
}
```

k.hpp file

```
#include x.hpp
```

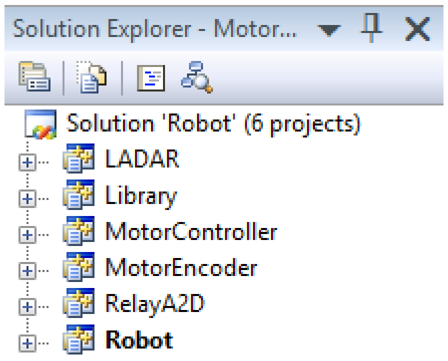
o.hpp file

```
#include x.hpp
#include h.hpp
```

This way, o.hpp would include x.hpp only once.

Startup Project

By default, startup projects are the ones that run automatically when Visual Studio debugger starts [3]. In the solution file Robot.sln, the startup project (Robot.vcproj) can be identified in the solution explorer window of visual studio by bold letters. See figure below.



Solution 'Robot' (6 projects)
- LADAR
- Library
- MotorController
- MotorEncoder
- RelayA2D
- **Robot**

Figure 34 . Solution Explorer Window.

This is the most important project file of the embedded software and is composed of main source files(.cpp) from other project files in addition to other files specific for this startup project. Notably, it does not contain test files from other projects. The class diagram for this project is:

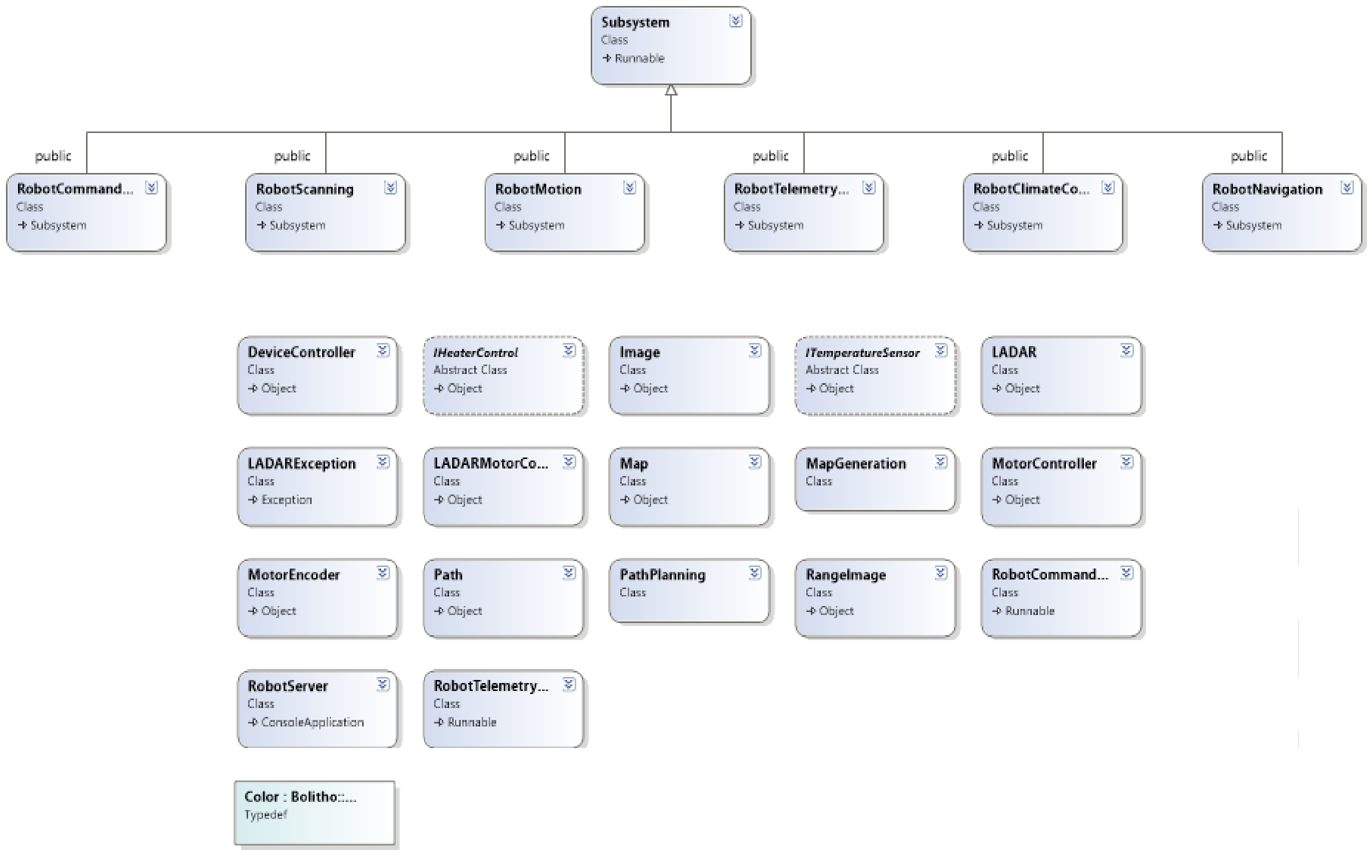


Figure 35. Class Diagram

As seen in the class diagram, the embedded system is composed of six main subsystems. The subsystems RobotCommand and RobotTelemetry are related to the connection between the robot and the client, the reception of commands to the robot and also the send of data. RobotMotion is responsible for the movement of the robot, which is divided in three types: Drive, Turn and Scan.

struct PathStepType

{ enum { DRIVE, TURN, SCAN }; };

The subsystem RobotMotion controls and manages the position of the robot, while RobotNavigation controls and manages the navigation system of the robot. Finally, the TemperatureControl subsystem would be responsible for controlling the temperature of certain targets and of certain subsystems of the robot. However, this feature was not actually implemented, since there are no temperature sensors in the robot hardware.

LADAR.vcproj

This project is composed of the following source files: LADARControl, LADARMotorControl, LADARTest, Map, and RangeImage. The source files in these project are responsible for the control and test of LADAR's stepper motor, in addition to LADAR's image gathering.

Library.vcproj

This project is responsible for providing important classes and methods that support the software as whole. For example, dealing with colors, date and time, and math.

MotorController.vcproj

In this project, the motor to be controlled is chosen, being either the ones from left or the ones in the right. Also, it initializes the motors, stop them and set their speed. There is also a MotorControllerTest.cpp file in this project.

MotorEncoder.vcproj

This project is composed of two source files (MotorEncoder.cpp and MotorEnconderTest.cpp), in addition to three header files (Config.hpp, MotorEncoder.hpp and Robot.hpp). This project is related to receiving the data from the encoder, testing them, and also comparing the data to the expected one. It is very important for the control system of the motors, since the CPU receives information regarding the displacement of the motors in a feedback from the encoders.

Conclusion

Studying both the controller software (placed on the client's computer) and the embedded system software (placed on the robot) is extremely important in order to understand how they communicate, allowing data transfer and the control of the robot. Furthermore, this report can be studied in parallel with the hardware one. This way, future teams who work on Nanook will have better understanding of how hardware and software interface and have an easier start, allowing the development of further applications.

References

[1] *https://msdn.microsoft.com/en-us/library/2208a1f2.aspx*

[2]http://stackoverflow.com/questions/7133796/what-are-sln-and-vcproj-files-and-what-do-they-contain

*[3] https://msdn.microsoft.com/en-us/*library/*26k97dbc(v=vs.90).aspx*