Deliverable 3

Project Testing and Delivery Document

C-Lab

Ashwath George	9733469
Nicholas Gignac	9128964
Robert Jakubowicz	6045707
Caroline Labbe	6320945
Brian Lam	1696785

For the course:
COMP354
Introduction to Software Engineering

Presented to:
Sutharsan Sivagnanam
August 14, 2013

Revision History

Revision	Date	Author(s)	Description
0.0	07/08/13	Caroline	Added Latex template
0.1	07/08/13	Nicholas	Added introduction
0.2	08/08/13	Brian	Added test coverage
0.3	09/08/13	Robert	Added unit test cases
0.4	10/08/13	Ashwath	Added requirements test scenarios
0.5	10/08/13	Nicholas	Added installation and user manual
0.6	11/08/13	Nicholas	Added stress testing
0.7	12/08/13	Nicholas	Added final cost estimate
0.8	12/08/13	Caroline	Fix latex formatting
0.9	12/08/13	Caroline	Insert figures at the right places
1.0	12/08/13	Caroline	Assemble all sections to complete document
1.1	13/08/13	Caroline	Fix some typos after review

Contents

List of Figures

List of Tables

1 Introduction

The C-Ker Vessel Monitoring System, from C-Lab, has been completed and the final iteration has been demoed successfully. This document will provide a detailed analysis of the various tests performed on the system prior to its submission, as well as the corresponding results subsequently obtained. An installation and user manual will also be included, as well as the final revision of the Cost Estimation of the systems development.

2 Testing Report

2.1 Test Coverage

2.1.1 Tested Items

The tested items can be categorized into requirements and units.

Tested Requirements

All application features are tested thoroughly. However, mandatory functionalities listed in the requirement specification document are treated with a higher priority.

- Listing Vessels in a Table

  This is a core feature which allows users to view vessel information in a list.

- Displaying Vessels on a Radar

  This is a core feature which allows users to visualize vessels graphically.

- Filtering Vessels by Type

  This is a core feature which allows users to view only specific vessel types.

- Sorting Vessels by Attribute

  This is a core feature which allows users to organize vessel information by data.

- Loading Vessel Scenario Files

  This is a core functionality which allows the application to load simulation properties.

- Simulating Vessel Movement

  This is a core functionality which allows the application to have moving vessels.

- Triggering Risk Alarms

  This is a core functionality which allows the application to display alarm alerts in both the table and the radar when vessels get too close to each other.

- Login Authentication

  This is a core functionality which allows users to login as either an administrator or an operator; only administrators can filter and sort vessels.

- Maximum of 100 Vessels in Simulation

  This is a mandatory non-functional requirement which allows the simulation to handle up to one hundred vessels at the same time.

- Shortest Time Step of 0.5 Seconds in Simulation

  This is a mandatory non-functional requirement which allows the simulation to update at a time step from 0.5 seconds and up.

Tested Units

The tested units include the ScenarioParser, the Authenticator, the Vessel, the VesselPresenter, and the LoginPresenter classes.

- ScenarioParser

  This class takes care of parsing and loading scenario files. It is important to test because the simulation entirely depends on the success of the file data being loaded correctly. In addition to reading the file accurately, it should also reject files that are not properly formatted.

- Authenticator

  This class takes care of authenticator user login attempts. It is important to test in order to ensure that user privileges are enforced correctly. Only a correct combination of username and password may access the application's main window.

- Vessel

  This class encapsulates vessel data and methods. It is important to test that the distance between two vessels is calculated correctly, so that alarms can be generated properly. It is also important test that the vessel positions are updated correctly, so that the simulation can run properly.

- VesselPresenter

  This class presents the vessel information to be displayed by the interface; it also takes care of sorting and filtering vessels. It is important to test because sorting and filtering are core features which are used fairly frequently by the user. The filtering function should remove unwanted vessel types from the list of vessels displayed, while the sorting function should order the list of vessels displayed by a specified attribute.

- LoginPresenter

  This class presents authentication information to the login screen. It is important to test in order to ensure correct functionality of the login screen. The correct user type should be identified properly when given a specific user.

2.1.2 Untested Items of Interest

The vessel data from the scenario files are not tested for realism or plausibility; it is accepted as long as the formatting is correct. In other words, it is entirely possible that certain vessel types go at very unrealistic speeds, such as a human going 340 meters per second. In order to reject unrealistic scenarios, further research on vessel types must be performed. It then becomes a simple matter of testing maximum and minimum value constraints. In our case, it is up to the user to provide a scenario that makes sense.

2.2 Test Cases

2.2.1 Unit Testing

ScenarioParser class

The ScenarioParser class is tested using white box testing since it has been developed by us. The tests are in the class ParserTest.cs which is run by NUnit. This class takes care parsing scenario files and extracting information that goes into the vessels list or details about the simulation.

ParseText (string text)

Condition: Text is empty

Description: This test inputs an empty string and checks that no vessel is returned.

```
    public void ParseText_EmptyText_ReturnsEmptyList()
{
    string text = "\n";
    List<Vessel> actual = ScenarioParser.ParseText(text);
    List<Vessel> expected = new List<Vessel>();
    Assert.AreEqual(expected.Count, actual.Count);
}
```

ParseText (string text)

Condition: Text is a valid vessel line

Description: This test inputs a string with each field of a vessel and checks that each field on the returned vessel matches.

```
    public void ParseText_ValidLine_ReturnsListWithOneVesselObject()
{
    string text = "NEWT 001 1 4990 0 0.1 0.1 10\n";
    List<Vessel> actual = ScenarioParser.ParseText(text);
    List<Vessel> expected = new List<Vessel>();
    expected.Add(new Vessel(new string[]  "NEWT", "1", "1", "4990",
"0", "0.1", "0.1", "10" ));
```

```
    Assert.AreEqual(expected.Count, actual.Count);

    for (int i = 0; i != expected.Count; i++)
    {
    Assert.IsTrue(new VesselComparer().Equals(expected[i], actual[i]));
    }
}
```

ParseText (string text)

Condition: Text line is missing NEWT

Description: This test inputs a string with a missing NEWT at the beginning and checks that no vessel is returned.

```
    public void ParseText_InvalidLineMissing_ReturnsZero()
{
    string text = "001 1 4990 0 0.1 0.1 10\n";
    List<Vessel> actual = ScenarioParser.ParseText(text);
    int expected = 0;
    Assert.AreEqual(expected, actual.Count);
}
```

ParseText (string text)

Condition: Text contains an invalid value

Description: This test inputs a string where an numerical value is expected but a letter is given. It then checks that a FormatException is thrown.

```
public void ParseText_InvalidLineValue_ThrowsFormatException()
{
    Assert.Throws<System.FormatException>(
    delegate
    {
    string text = "NEWT 001 1 4990 a 0.1 0.1 10\n";
    ScenarioParser.ParseText(text);
    }
    );
}
```

ParseText (string text)

Condition: Text contains an undefined keyword

Description: This test inputs a string with the keyword NOTEXISTING and then checks that no vessel is returned.

```
    public void ParseText_InvalidLineKeyword_ReturnsZero()
{
    string text = "NOTEXISTING 001 1 4990 0 0.1 0.1 10\n";
    List<Vessel> actual = ScenarioParser.ParseText(text);
    int expected = 0;
    Assert.AreEqual(expected, actual.Count);
}
```

ParseText (string text)

Condition: Text contains details about the simulation: starttime, timestep, time and range and the value are integers.

Description: This test inputs a string with integer values for the simulation details. It then checks that the ScenarioParser.Simulator contains the correct values.

```
    public void ParseText_ValidSimulationIntegerDetails_ReturnsFilledScenari
{
    string text = "STARTTIME 0\n" + "TIMESTEP 1\n" + "TIME 500\n"
+ "RANGE 500\n" ;
    ScenarioParser.ParseText(text);
    float expectedStartTime = 0f;
    float expectedTimeStep = 1f;
    float expectedTime = 500f;
    int expectedRange = 500;
    Assert.AreEqual(expectedStartTime, ScenarioParser.Simulator.StartTime,
float.Epsilon);
    Assert.AreEqual(expectedTimeStep, ScenarioParser.Simulator.TimeStep,
float.Epsilon);
    Assert.AreEqual(expectedTime, ScenarioParser.Simulator.Time,
float.Epsilon);
    Assert.AreEqual(expectedRange, ScenarioParser.Simulator.Range);
}
```

ParseText (string text)

Condition: Text contains details about the simulation: starttime, timestep, time and range and the value are floats.

Description: This test inputs a string with float values for the simulation details. It then checks that the ScenarioParser.Simulator contains the correct values.

```
    public void ParseText_ValidSimulationFloatDetails_ReturnsFilledScenarionI
{
    string text = string text = "STARTTIME 0.3\n" + "TIMESTEP 1.1\n"
+ "TIME 500.5\n" + "RANGE 500\n";
    ScenarioParser.ParseText(text);
    float expectedStartTime = 0.3f;
    float expectedTimeStep = 1.1f;
    float expectedTime = 500.5f;
    int expectedRange = 500;
    Assert.AreEqual(expectedStartTime, ScenarioParser.Simulator.StartTime,
float.Epsilon);
    Assert.AreEqual(expectedTimeStep, ScenarioParser.Simulator.TimeStep,
float.Epsilon);
    Assert.AreEqual(expectedTime, ScenarioParser.Simulator.Time,
float.Epsilon);
}
```

ParseText (string text)

Condition: Text contains details about the simulation that are invalid.

Description: This test inputs a string of integer values for the simulation details where some values are letters instead of numeric. It then checks that a FormatException is thrown.

```
    public void ParseText_InvalidSimulationDetails_ThrowsFormatException()
{
    string text = "STARTTIME 0\n" + "TIMESTEP A\n" + "TIME B\n" +
"RANGE 500\n";
    Assert.Throws<System.FormatException>(
    delegate
    {
    ScenarioParser.ParseText(text);
    }
    );
}
```

Results

Figure 1 shows the results as run from NUnit. It shows all the tests passing.

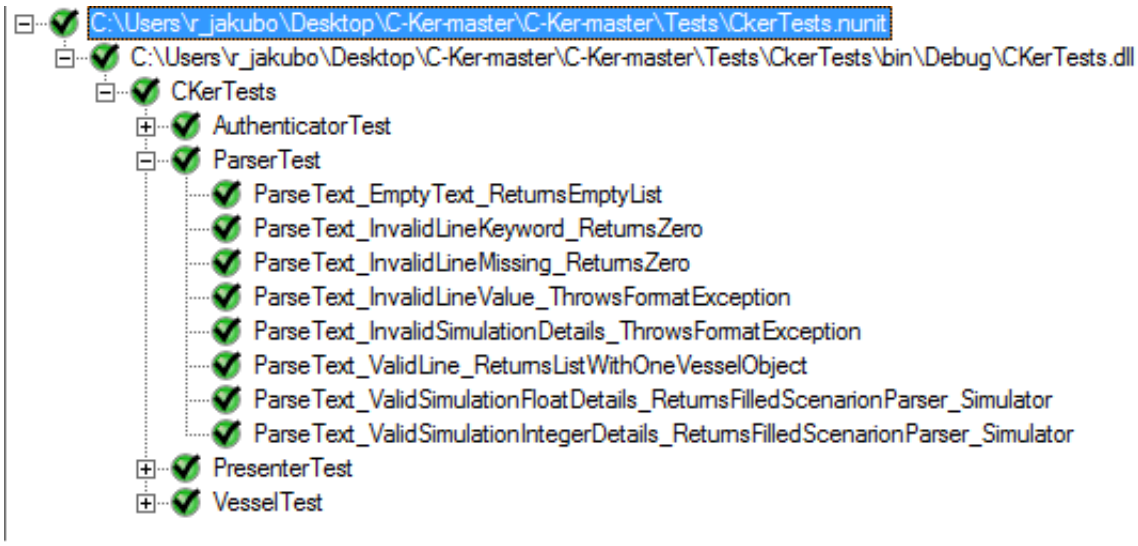

Figure 1: ScenarioParser class

Authenticator class

The Authenticator class is tested using white box testing since it has been developed by us. The tests are in the class AuthenticatorTest.cs which is run by NUnit. This class takes care of correctly identifying valid user and rejecting invalid ones. Hence, different username and password combinations must be thoroughly tested.

FindUser(string username)

Condition: Invalid username

Description: This test inputs an invalid username and check that no user is returned.

```
    public void FindUser_InvalidUsername_ReturnsNull()
{
    string username = "InvalidUsername";
    User actual = Authenticator.FindUser(username);
    Assert.IsNull(actual);
}
```

FindUser(string username)

Condition: Valid username

Description: This test inputs a valid username and check that a valid user object is returned.

```
public void FindUser_ValidUsername_ReturnsInstanciatedUserObject()
{
    string username = "admin";
    User expected = new User();
    expected.Name = username;
    User actual = Authenticator.FindUser(username);
    Assert.AreEqual(expected.ToString(), actual.ToString());
}
```

FindUser(string username, string password)

Condition: Invalid username and password

Description: This test inputs an invalid username and password and check that no user is returned.

```
public void FindUser_InvalidUsernameAndPassword_ReturnsNull()
{
    string username = "InvalidUsername";
    string password = "InvalidPassword";
    User actual = Authenticator.FindUser(username, password);
    Assert.IsNull(actual);
}
```

FindUser(string username, string password)

Condition: Valid username and password

Description: This test inputs a valid username and password check that a valid user object is returned.

```
public void FindUser_ValidUsernameAndPassword_ReturnsInstanciatedUserObject(
{
    string username = "admin";
    string password = "fullaccess";
    User expected = new User();
    expected.Name = username;
    expected.Password = password;
    User actual = Authenticator.FindUser(username, password);
    Assert.AreEqual(expected.ToString(), actual.ToString());
}
```

Logout()

Condition: A user is logged in

Description: This tests logs in a valid user and check that no user is logged in after logout() is called.

```
    public void Logout_AUserLoggedIn_ReturnsNull()
{
    Authenticator.Login("admin", "fullaccess");
    Authenticator.Logout();
    Assert.IsNull(Authenticator.CurrentUser);
}
```

Login(string username, string password)

Condition: Invalid username and password

Description: This tests inputs an invalid username and password and tries to login that user. Since they are invalid, it checks that no user is logged in.

```
    public void Login_InvalidUser_ReturnsCurrentUserNull()
{
    Authenticator.Login("adminfake", "fullaccessfake");
    Assert.IsNull(Authenticator.CurrentUser);
}
```

Login(string username, string password)

Condition: Valid username and password

Description: This tests inputs a valid username and password and tries to login that user. Since they are valid, it checks that a valid user is logged in.

```
    public void Login_ValidUser_ReturnsMatchingUsernameAndPassword()
{
    Authenticator.Login("admin", "fullaccess");
    string expected = "admin";
    string actual = Authenticator.CurrentUser.Name;
    Assert.AreEqual(expected, actual);
    expected = "fullaccess";
    actual = Authenticator.CurrentUser.Password;
    Assert.AreEqual(expected, actual);
}
```

Results

Figure 2 shows the results as run from NUnit. It shows all the tests passing.

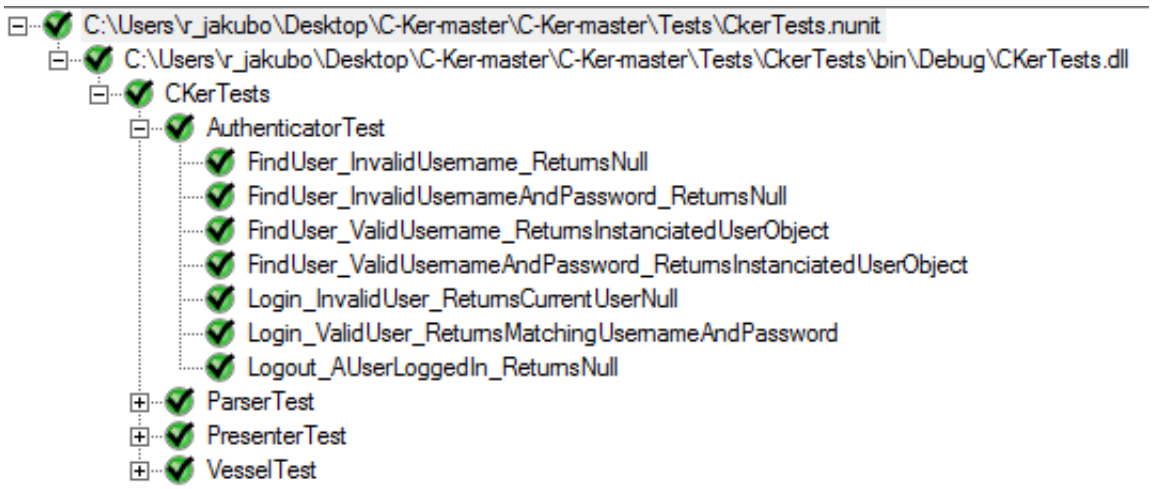

Figure 2: Authenticator class

2.2.2 Requirements Testing

This section contains a table (table 2 & 3) that elaborates on the tests that we performed in order to validate the various use cases implemented in the final iteration of the C-Ker Vessel Monitoring System.

Case	Input	Expected Result	Result
1. Listing Vessels in a Table			
Default scenario file "comp354_vessel.vsf" is chosen	Load "comp354_vessel.vsf" into simulator	Vessels within range are listed in the table, and are added and deleted as they move in and out of range	OK
2. Displaying Vessels on a Radar			
Default scenario file "comp354_vessel.vsf" is chosen	Load "comp354_vessel.vsf" into simulator	Vessels within range are displayed on the radar, and disappear and appear as they move in and out of range	OK
3. Filtering Vessels by Type			
Unfiltered table/radar view contains vessel type "Human"	Uncheck "Human" checkbox	"Human" vessels are no longer displayed in table/radar view	OK
Unfiltered table/radar view does not contain vessel type "Human"	Uncheck "Human" checkbox	No change in table/radar view	OK
4. Sorting Vessels by Attribute			
Unsorted/sorted table contains vessels with unique "Vessel ID"	Click on "Vessel ID" field once	Table sorts vessels by descending order according to "Vessel ID"	OK
Unsorted/sorted table contains vessels with unique "Vessel ID"	Click on "Vessel ID" field twice	Table sorts vessels by ascending order according to "Vessel ID"	OK
5. Loading Vessel Scenario Files			
Loading a scenario file with bad values	Load "scenario_badvalues.vsf" into simulator	Error message "File has invalid formatting." Is displayed	OK
Loading an empty scenario file	Load "scenario_empty.vsf" into simulator	Error message "File has invalid formatting." Is displayed	OK

Table 2: Requirements Testing Part 1

Case	Input	Expected Result	Result
6. Simulating Vessel Movement			
A scenario file containing vessels with non-zero X and/or Y velocities is chosen	Load "scenario_20vessels.vsf" into simulator	Vessel blips move across the radar and corresponding vessel coordinates change in the table over a regular interval	OK
7. Triggering Risk Alarms			
A scenario file containing vessels with trajectories that intersect/are close to each other is chosen	Load "scenario_2.vsf" into simulator	The table entries for the corresponding vessels turn red/yellow, and the alarm animations are displayed on the radar	OK
8. Login Authentication			
A valid administrator username is entered in the login screen	Enter "admin" in the first field	The user avatar changes from the default image to the administrator image	OK
A valid administrator username is entered, but is followed by a space (depicted by '_')	Enter "admin_" in the first field	There is no change in the user avatar	OK
An invalid password/username or combination of the two is entered	Enter "admin" in the first field, and "derp" in the second	Error message "Invalid credentials; please try again." is displayed	OK
9. Maximum of 100 Vessels in Simulation			
A scenario file containing 100 vessels is chosen	Load "scenario_100vessels.vsf" Into simulator	All vessels in the scenario that are within the scenarios range are displayed in the table/radar view	OK
10. Shortest TIMESTEP of 0.5 Seconds in Simulation			
A scenario file containing a TIMESTEP of 0.5 is chosen	Load "scenario_10delay.vsf" into simulator	The table and radar views update twice every second	OK

Table 3: Requirements Testing Part 2

2.2.3 Stress Testing

In the original Requirement Document, it was required for the system to support up to 100 ships. As basic stress test, we conceived a simulation file with a thousand ships. Surprisingly enough, the C-Ker can run it without any difficulties. The Radar display is obviously very crowded and almost unreadable, but it still functions accordingly. For the purpose of this simulated stress test, we will also push other limits, such as range, speeds, refresh rate and simulation duration. We know for a matter of fact, after all the previously mentioned tests, that within reasonable conditions, the C-Ker VMS functions properly. First, we could stress test parameters individually. First, push the range of the radar from the current 5,000 meters cap, up to 50,000 meters, and run a standard 100 ship simulation until all ships have left the radars range. Then, we could push the limits of the boat speeds to close-to jet-speed speeds, thus testing the efficiency of the refresh rate, followed by an increase in the refresh rate, to add additional stress on the computing aspect of the system. And finally, we could test de duration efficiency of the system by designing a simulation with a duration time above 24 hours.

Ultimately, if all of the individual stress test mentioned above pass without resulting in a system crash or else, we could throw one, last, ultimate stress test at the C-Ker, which would implicate the joining of all previously, mentioned individual parameter stress tests. This ultimate challenge simulation could be based on 2,000 vessels and more, up to 10,000 vessels, all in a timed pattern to ensure the presence of at least 200 vessels at any given time within the radar range, which itself would be increased to above 10,000 meters. A given number of those vessels could be given excessively high speed rates while the radar refresh rate could be lowered in order to refresh more often. All of this, with a timing configuration to ensure the presence of the said 200 vessels at anytime for about one week of run-time (168 hours). This should prove to be the ultimate test for any VMS of the scale and scope of the C-Ker.

3 System Delivery

3.1 Installation Manual

Step 1: Downloading the system.

First, head to `https://github.com/Darkneon/C-Ker` to reach our download page. There you will find C-Labs repository for the C-Ker VMS. On the bottom right of this page, you will find a button identified as Download ZIP (see figure 3). Click it to start the download of your software.

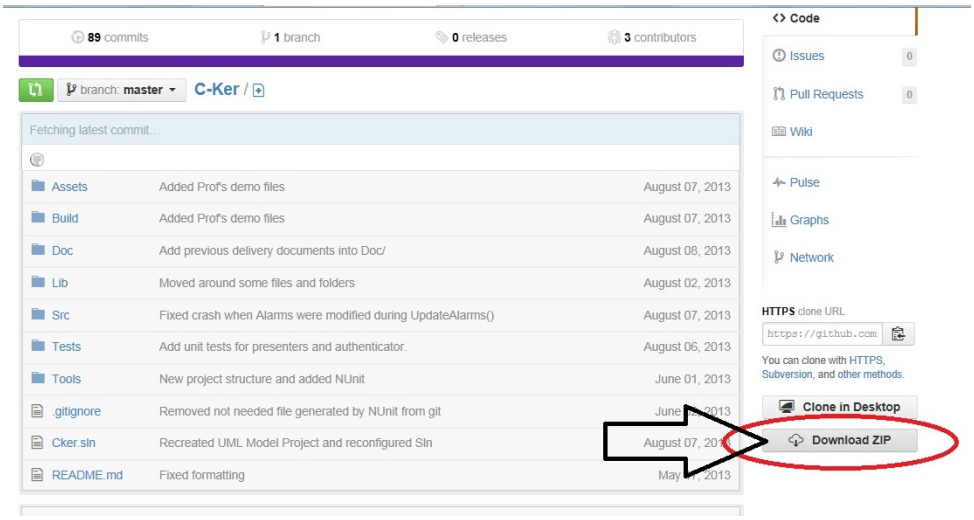

Figure 3: Download ZIP

Once you have located and clicked on the previously mentioned button, a box should prompt you to take action, by either Opening or Saving the ZIP file, or cancelling the operation. It is strongly recommended to Save the ZIP. More so, you should save it in a location of your choice by clicking on the smaller arrow next to the Save button on the prompt box (see figure 4), which will expose a Scroll box from which you should select Save as. From Save as, select the location of your choice for the C-Ker master ZIP file.

Depending on your browser and its version, the prompt box you see may be different than the one you currently see in the image below. If you are experimenting difficulties with the download prompt box of your web browser, you are strongly recommended to visit your browsers manufacturer instruction manual.

Figure 4: Save

Once you have saved the master ZIP file to the location of your choice, you may now proceed to the next step.

Step 2: Unzipping the Master File

Now that you have saved the master ZIP file at the location of your choice, locate it in its folder using Windows Explorer. If you cannot locate the ZIP file, repeat step 1 and try another location that could be easier for you to find. Once you have located the ZIP file (see figure 5), double left click it in order to open it.

Once you have double left clicked on the ZIP file, you should see the following screen opening (see figure 6), if not try again.

Once you have the ZIP file opened (figure 6), left click once on the button identified Extract to. It will open a different window to ask you where you wish to extract the files (see figure 7). If the new window does not open, click again.

Once the new window is opened, you will have the possibility to select the target location of your choice, either by typing the address (green box in figure 7) or using the explorer display (yellow box in figure 7). Once you are satisfied with your target location,

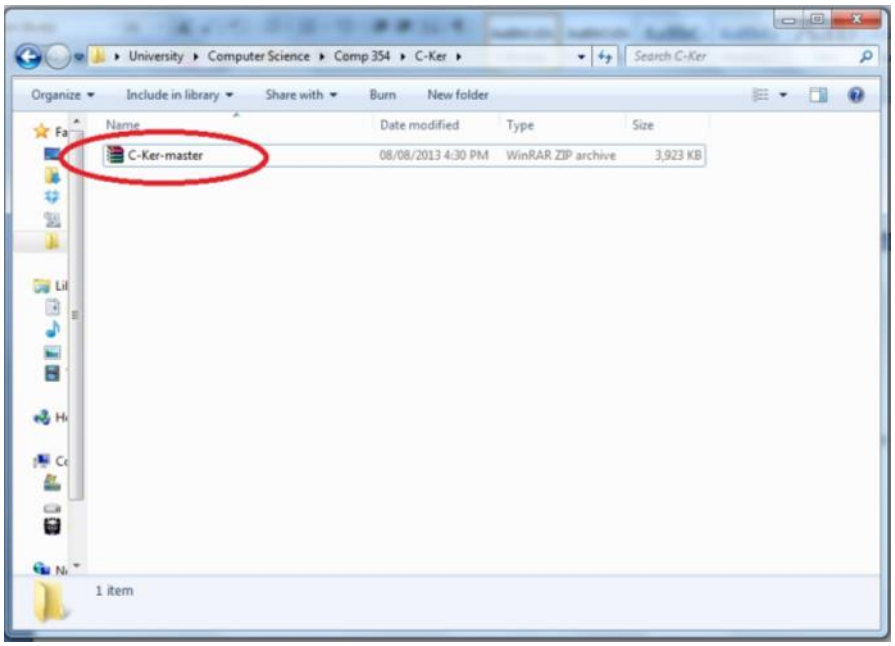

Figure 5: Open ZIP file

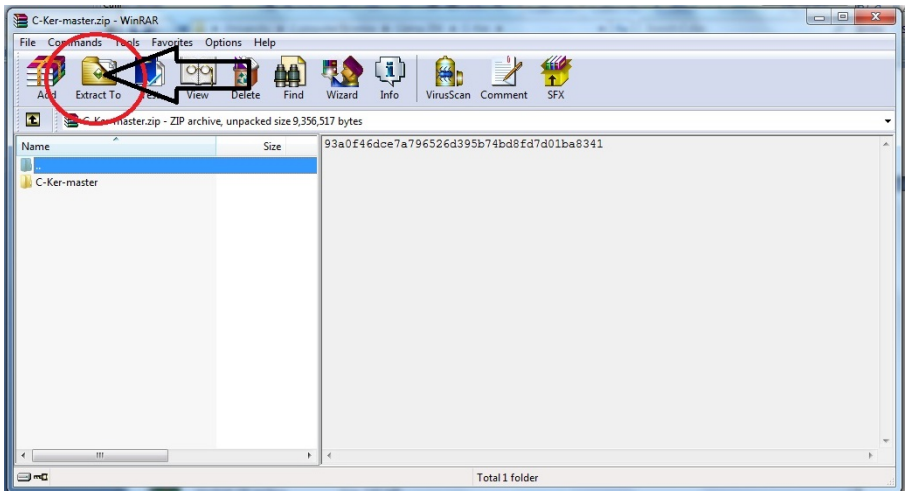

Figure 6: ZIP file

click on the OK button (red circle in figure 7). This will extract the ZIP file to the folder you selected, and create a new sub folder called C-Ker-master (see figure 8).

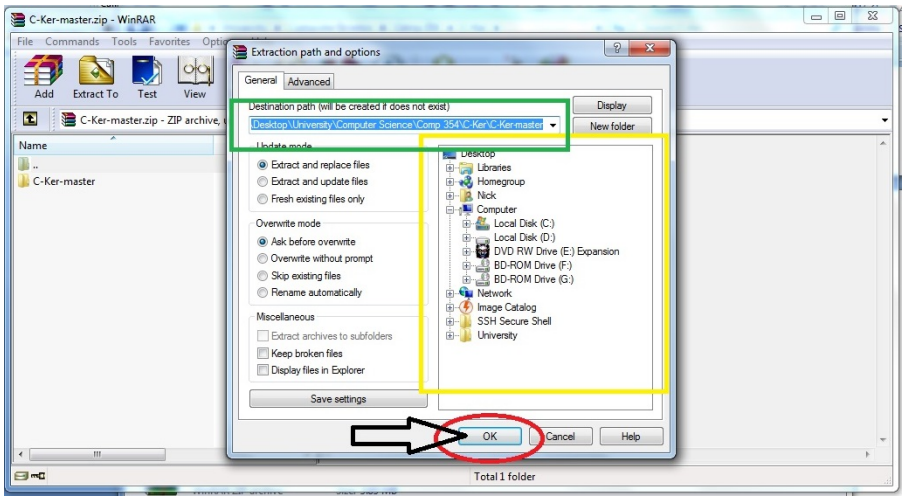

Figure 7: Extract file

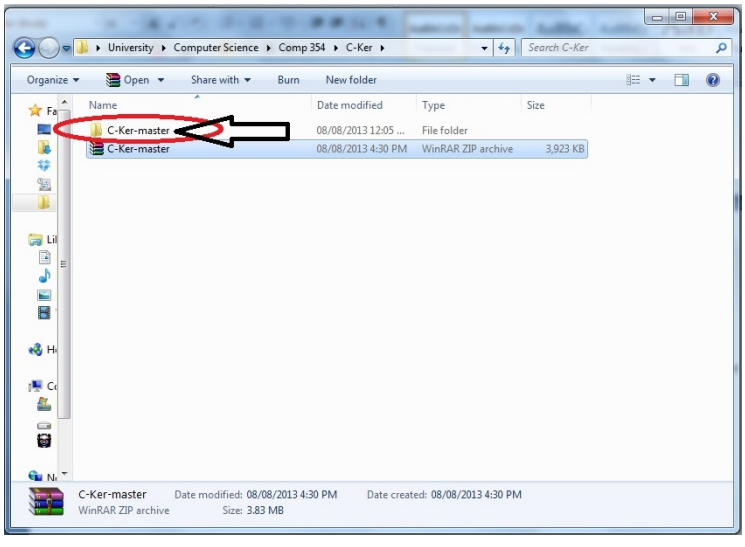

Figure 8: C-Ker-master

If you can see the folder, then you have successfully unzipped the file and are ready to proceed to the next step.

Step 3: Running the VMS.

To reach the executable file of C-Ker, start by double clicking on the new folder created in step 2 in order to access its content. Once it is open, you should have the view as seen in figure 9.

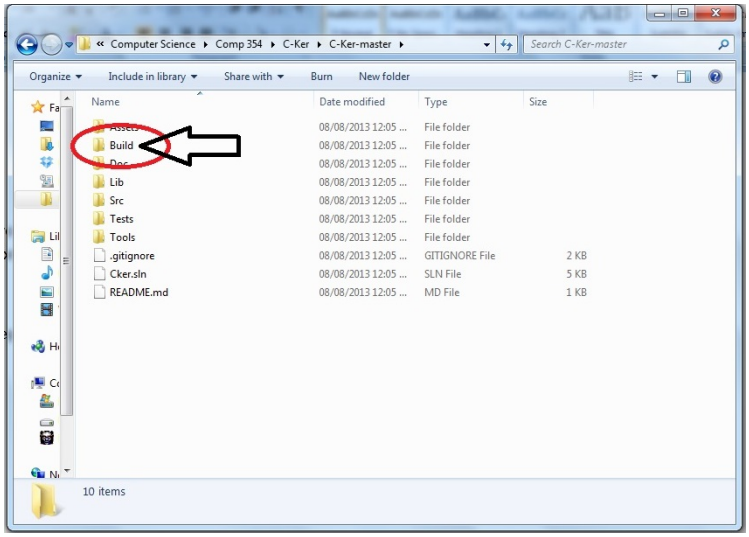

Figure 9: Openned C-Ker-master folder

Now that you have accessed the content of the C-Ker Master folder, locate the Build folder and double click it to open it.

Once this is done, locate the file named CkerGUI (pointed at by black arrow in figure 10) and double click it to launch the C-Ker VMS.

NOTE: Your computer may be running on a previous version of .NET Frameworks. If you receive an error message when trying to launch CkerGUI, head to the following hyperlink and follow the instructions to update your computers .NET Framework.

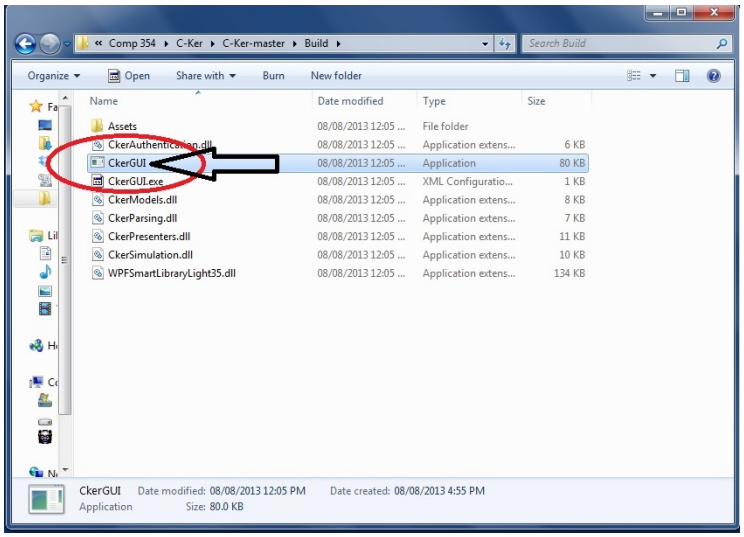

Figure 10: CKerGUI

`http://dotnetsocial.cloudapp.net/GetDotnet?o1=.NETFramework,Version=v4.5`

If everything works accordingly, you should see what is in figure 11 appears in the screen.

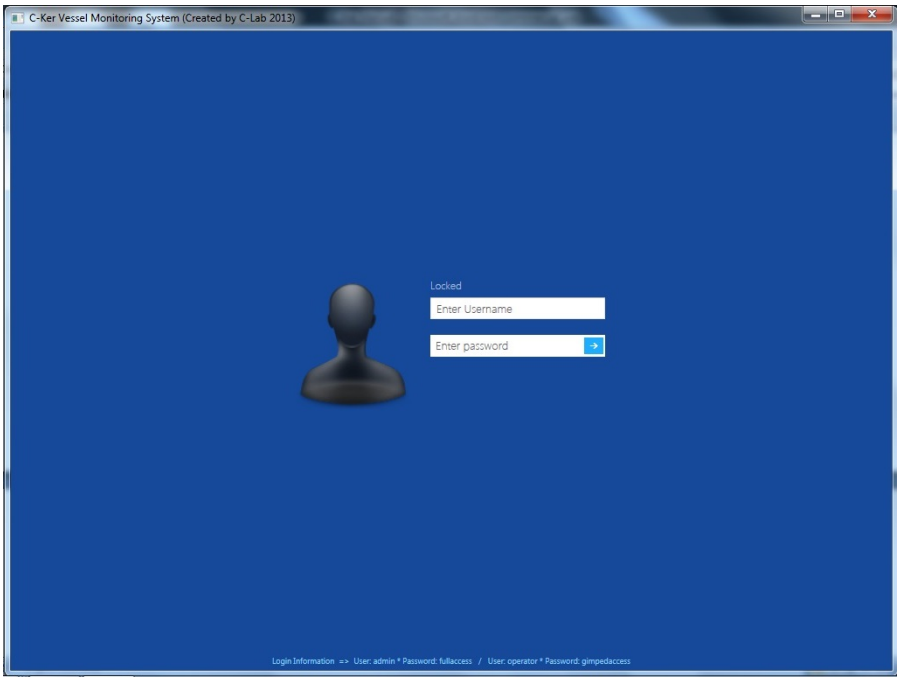

Figure 11: CKer login screen

3.2 Users Manual

In both of the following cases, the first, unmentioned, step is to launch the system. If you need instruction on how to launch C-Ker, refer to 1. Installation Manual Step 3.

(a) Operator Instructions

As C-Ker Operator type user, enter operator as Username and gimpedaccess as password. Forgot your password and/or Username? They are on display at the bottom of the log in screen (see figure 12).

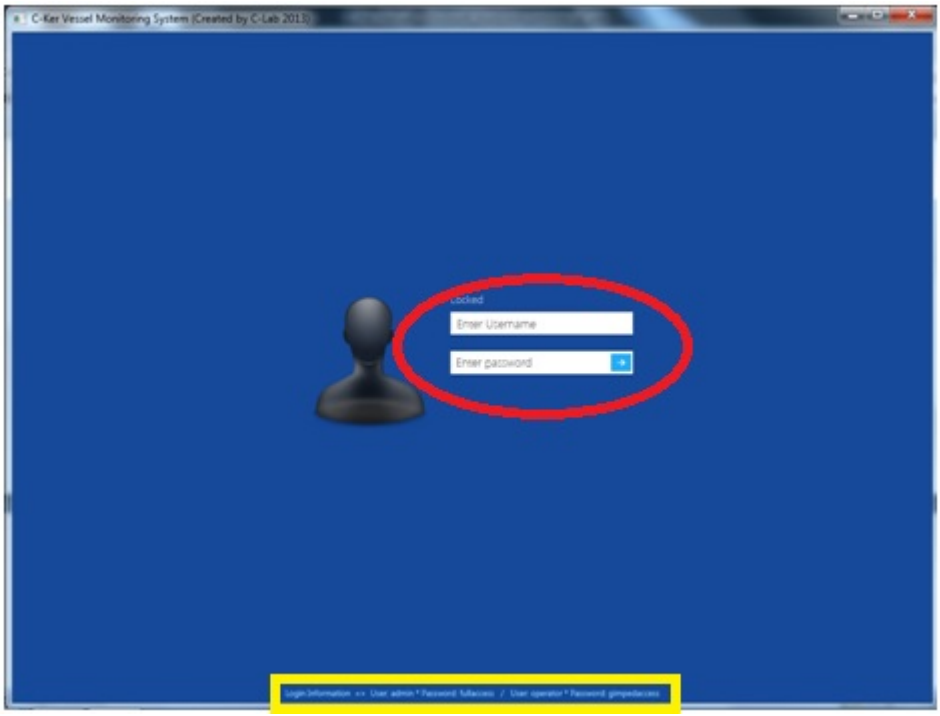

Figure 12: Logging In

Once you have completed your login, you should have access to the main page, which at this point should be blank. In order to load a simulation, click on the New Simulation button at the top left corner of the system (red square in figure 13), and a drop down box will appear with a list of preloaded simulations (yellow box in figure 13). If you wish to import and/or run a simulation from another, not preset, location, click

on the Browse button (green square in figure 13), and a standard Windows Explorer file loader menu will open. Once you have found the simulation file of your choice, either in the preset menu or Browse section, simply double click on the file to open it and load the simulation onto the VMS. Once this is done, your simulation should be playing and C-Ker should be displaying something similar to the figure 14. If you wish to load a new simulation simply repeat the step mentioned before and the new simulation will immediately replace the current one. To quit or change log in type, exit the system using the red X at the top right corner of the software, like any other software. For a full glossary of the icons displayed on the C-Ker, jump to section c) Glossary & References.

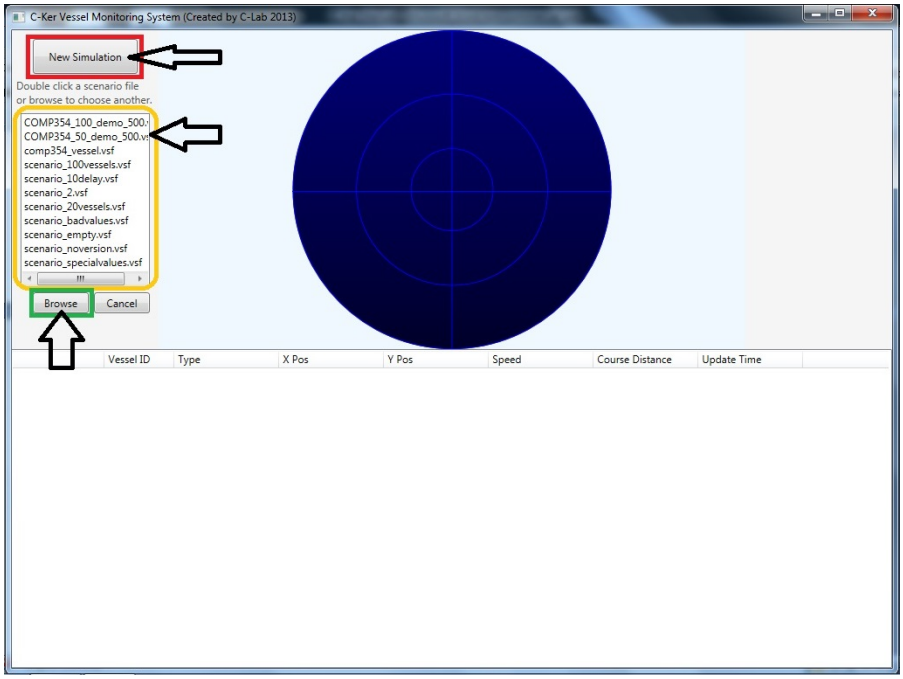

Figure 13: New Simulation

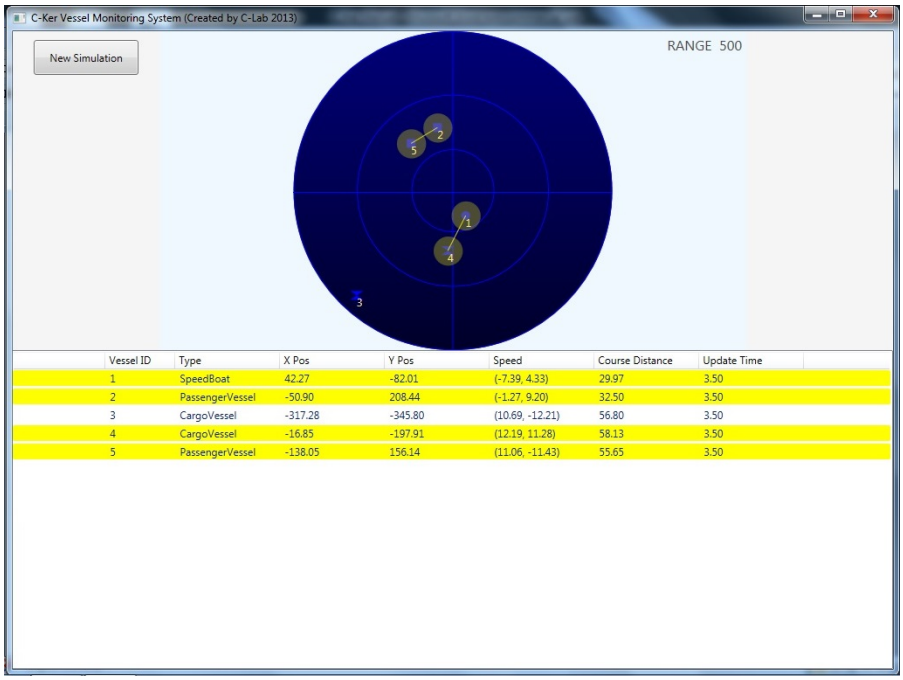

Figure 14: Operator View

(b) Administrator Instructions

The log in instruction for Administrator access is the same as for an Operator access. Simply use the Username admin and password fullaccess to gain access to the C-Ker in Administrator mode. The loading of a simulation uses the same procedure as for the Operator access. Once the selected simulation has loaded, you will gain access to two features unavailable in the Operator access: Sorting and Filtering vessels.

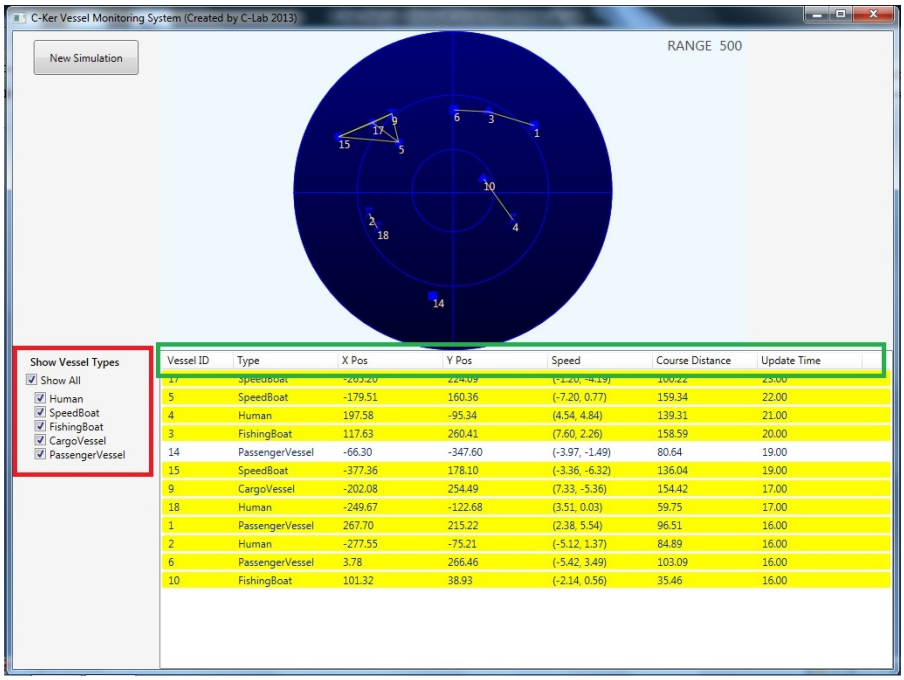

Figure 15: Administrator View

Vessel Filtering:

In order to filter vessels, use the checkboxes at the left of the tab (red box in figure 15). If you wish to hide/show all vessels, click the checkbox names Show All. It will either hide or show all vessels both on the radar display and tab. If you wish to either hide or show a specific type of vessel, check or uncheck its corresponding checkbox.

Vessel Sorting:

In order to sort vessels in the tab display, click on the parameters column by which you wish to sort the vessels (green box in figure 15). Depending on the current sorting method of the tab, click once on the parameters column of your choice will sort the vessels from smallest to largest parameter value, and twice will sort it from largest to lowest parameter value. Note that all 7 columns can be sorted.

(c) Glossary & References

Glossary

- Human (swimmer)

  Represented by the hollow-X shape (figure 16)

Figure 16: Human (swimmer)

- Fishing Boat

  Represented by the Triangle (figure 17)

Figure 17: Fishing Boat

- Speed Boat

  Represented by the Circle (figure 18)

Figure 18: Speed Boat

- Cargo Ship

  Represented by Hourglass (figure 19)

Figure 19: Cargo Ship

- Passenger Ship

  Represented by the Square (figure 20)

Figure 20: Passenger Ship

Reference

The following reference section will provide additional information and clarification about items not shown prior.

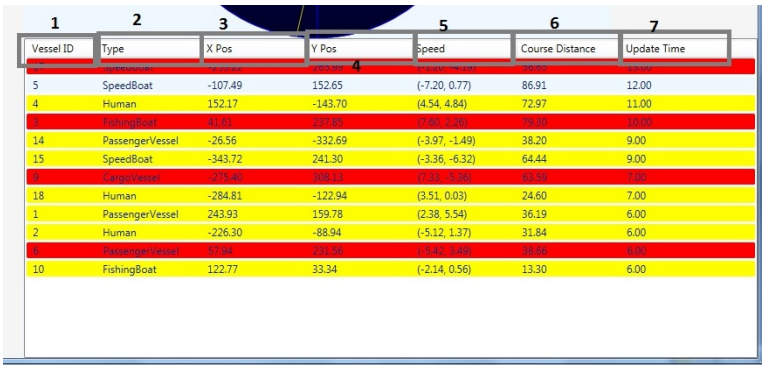

Figure 21: Tab Display

Tab Display (figure 21):

(1) Vessel ID:

The number ID assigned to the ship

(2) Type:

Type of vessel

(3) X Pos:

Location on the East-West X axis

(4) Y Pos:

Location on the North-South Y axis

(5) Speed:

Velocity at which the vessel is currently traveling

(6) Course Distance:

Distance from Radar Origin (0, 0)

(7) Update Time:

Total time since vessel has appeared on the VMS

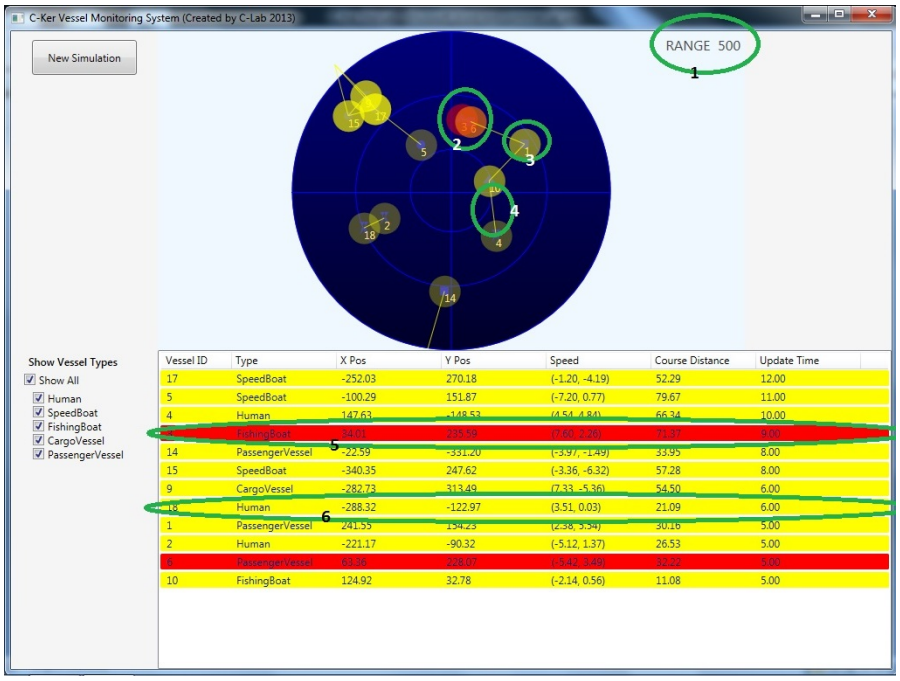

Figure 22: Radar Display

Radar display (figure 22):

(1) Range

The Range at which the radar is set. Equivalent to the radius of the circle of the radar

(2) Glowing Red Circle

Indicates the ship at the center of the circle is in high danger of collision, and is within 50 meters of another ship

(3) Glowing Yellow Circle

Indicated the ship at the center of the circle is potentially in danger of collision, and is between 50 and 200 meters away from another ship

(4) Yellow Line

Indicates the relationship between two ships in mutual Yellow Alert

(5) Red Tab Line

Indicates, within the tab display, that a ship is in high danger of collision, same as 2- Glowing Red Circle

(6) Yellow Tab Line

Indicated, within the tab display, that a ship is potentially in danger of collision, same as 3- Glowing Yellow Circle

4 Final cost estimate

With the delivery of the completed product comes the time to assess the cost associated with the development of the said product. In order to pinpoint as accurately as possible the total cost of the development of the C-Ker VMS, we will evaluate the following points.

4.1 Technical Resources

Since Delivery 1, our technical resource estimation and consumption has not changed, except for maybe a language library. We have not changed anything, compiler software, repository or language. Since everything we use was free, either a free service or software licensed via ENCS, we cannot account for any costs.

4.2 Time/Human Resources Cost

The entire cost evaluation and calculation of the project resides on the time and human resources consumed for the realisation of the C-Ker. As most hours worked on an item or another were registered by the members of C-Labs into a time log, we are capable of safely estimating the exact cost in hours of the development process.

Generally speaking, the documentation task for the delivery of this project used roughly 50% of the time put into the project by the members of C-Labs. According to the time logs of each member, about 320 hours of man-time went into this project, spread across our 5 members. We consider the time logs only as indicators and not accurate counter, as the proper use of the time log was not implemented early in the project, but rather just in time for delivery 3. The programmers logged a few less hours than the documentation/administration team, most likely because they did not consider the meeting times into their logs. It is our believe that everyone on C-labs has worked more hours than stipulated in time logs, but since this is only a cost estimate and not an accurate cost calculation, we will work with those numbers.

Ash and Brian had similar time consumption percentage and a similar total of hours, both been around 40-50 hours spent, with about 50% of their time dedicated to coding/programming or similar functions, and 50% of their time documenting. On the other hand, Robert, with his technical leader responsibilities, spent about 60 hours split

in three, roughly even, parts: development, deployment and designing, the last which includes documentation for the designs.

On the contrary, Nick and Caros time were above that of Roberts 60 hours, and were mostly dedicated to documenting, note taking and team coordination. They did spent some time testing and running the program in order to help the programming team with the testing phase of the delivery.

4.3 Incalculable Costs

Considering this is a non-complete Cost Evaluation based on a school project, many costs items had to be left behind unconsidered and uncalculated. For a starter, even if the Human Resources consumption was calculated in hours, wages are not considered in our evaluation. Also, since we used the optimal programming language that suited our main programmers best, we did not have to waste any resources on training on new technical items. Furthermore, everything regarding overhead expenditures, such as rent, electricity, and insurance, is ignored. For the hardware and software costs, we are also negating those as we did not purchase anything for this project, therefore, we will also negate the amortisation on the cost. Also, we were not able to reuse code from previous project, as none of us had worked on a VMS before. Exception made for a few items for the log in system, but not enough to be noticeable. In the same line of thought, we will not be able to reuse most of the code from the C-Ker since it is too specialized, and thus cannot be calculated as recovered cost.

4.4 Risk Induced Costs

This project was blessed with a minimal amount of risk factor actually happening. Most of the duration of the project went without any major issues but one noticeable incident. One team member had health issues within her family, and had to miss a few team meetings and could not provide the same amount of work as plan. Thankfully, she managed to get her section done in time as the rest of the team picked up the slack, noting the Meeting Minutes and so on, which resulted in very little impact on the final result. Also, it is fair to say that every team member had his off moment at a time or another, where emails were not returned promptly, or the level of implication was not to the expected level. In the end, not only did it had no effect on the final result, but the team was proficient enough that no team member ever needed to be brought back in line or asked to raise the level of production.