Team Autono-Mo

Jacobia

Detailed Design Specification

Team Members:
Bill Butts
Darius Salemizadeh
Lance Storey
Yunesh Shakya

Last Updated: 22 July 2012 @ 8:29:00 PM
# Table of Contents

List of Figures ........................................................................................................... vii  
List of Tables ............................................................................................................... viii  
Introduction ............................................................................................................... 1  
  Overview ................................................................................................................... 1  
  Purpose ...................................................................................................................... 1  
  Project Scope ............................................................................................................ 1  
Architecture Overview ............................................................................................... 2  
  General ...................................................................................................................... 2  
  Layer-Name and Description .................................................................................... 2  
    Sensory Layer ......................................................................................................... 2  
    Navigation Layer .................................................................................................... 3  
    Application Layer .................................................................................................. 3  
    Movement Layer .................................................................................................... 3  
Inter-Subsystem Dataflow .......................................................................................... 4  
  Overview ................................................................................................................... 4  
  Data Flows ................................................................................................................ 5  
  Producer-Consumer Relationships .......................................................................... 6  
  Layer-Level Producer-Consumer Relationships ................................................... 8  
Module Decomposition .............................................................................................. 10  
Sensory Layer Design ............................................................................................... 10  
  Overview ................................................................................................................... 10  
  Android GPS Sensor .............................................................................................. 11  
    Prologue ................................................................................................................ 11  
    Interfaces .............................................................................................................. 11  
  External Data Dependencies .................................................................................... 11  
  Internal Data Dependencies .................................................................................... 11  
  Process/Pseudocode .............................................................................................. 11  
Ultrasonic Sensor Array ............................................................................................. 11  
  Prologue ................................................................................................................... 11  
  Interfaces ................................................................................................................. 11
<table>
<thead>
<tr>
<th>Activity</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Data Dependencies</td>
<td>11</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>11</td>
</tr>
<tr>
<td>Process/Pseudocode</td>
<td>11</td>
</tr>
<tr>
<td>Android Compass Sensor</td>
<td>12</td>
</tr>
<tr>
<td>Prologue</td>
<td>12</td>
</tr>
<tr>
<td>Interfaces</td>
<td>12</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>12</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>12</td>
</tr>
<tr>
<td>Process/Pseudocode</td>
<td>12</td>
</tr>
<tr>
<td>getSensorData</td>
<td>12</td>
</tr>
<tr>
<td>Prologue</td>
<td>12</td>
</tr>
<tr>
<td>Interfaces</td>
<td>12</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>13</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>13</td>
</tr>
<tr>
<td>Process/Pseudocode</td>
<td>13</td>
</tr>
<tr>
<td>sendSensorData</td>
<td>13</td>
</tr>
<tr>
<td>Prologue</td>
<td>13</td>
</tr>
<tr>
<td>Interfaces</td>
<td>14</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>14</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>14</td>
</tr>
<tr>
<td>Process/Pseudocode</td>
<td>14</td>
</tr>
<tr>
<td>getGPS()</td>
<td>14</td>
</tr>
<tr>
<td>Prologue</td>
<td>14</td>
</tr>
<tr>
<td>Interfaces</td>
<td>14</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>14</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>14</td>
</tr>
<tr>
<td>Process/Pseudocode</td>
<td>14</td>
</tr>
<tr>
<td>getCompass</td>
<td>15</td>
</tr>
<tr>
<td>Prologue</td>
<td>15</td>
</tr>
<tr>
<td>Interfaces</td>
<td>15</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>15</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>15</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>15</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Navigation Layer Design</td>
<td>16</td>
</tr>
<tr>
<td>Localization Calculation</td>
<td>16</td>
</tr>
<tr>
<td>Prologue</td>
<td>16</td>
</tr>
<tr>
<td>Interfaces</td>
<td>16</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>17</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>17</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>17</td>
</tr>
<tr>
<td>Instruction Handler</td>
<td>17</td>
</tr>
<tr>
<td>Prologue</td>
<td>17</td>
</tr>
<tr>
<td>Interfaces</td>
<td>17</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>17</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>17</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>17</td>
</tr>
<tr>
<td>Path Calculation</td>
<td>18</td>
</tr>
<tr>
<td>Prologue</td>
<td>18</td>
</tr>
<tr>
<td>Interfaces</td>
<td>18</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>19</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>19</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>19</td>
</tr>
<tr>
<td>Threat Calculation</td>
<td>20</td>
</tr>
<tr>
<td>Prologue</td>
<td>20</td>
</tr>
<tr>
<td>Interfaces</td>
<td>20</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>20</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>20</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>20</td>
</tr>
<tr>
<td>Xbee Wireless Communicator</td>
<td>21</td>
</tr>
<tr>
<td>Prologue</td>
<td>21</td>
</tr>
<tr>
<td>Interfaces</td>
<td>21</td>
</tr>
<tr>
<td>External Data Dependencies</td>
<td>22</td>
</tr>
<tr>
<td>Internal Data Dependencies</td>
<td>22</td>
</tr>
<tr>
<td>Process/Pseudo code</td>
<td>22</td>
</tr>
<tr>
<td>Facilitator 2</td>
<td>22</td>
</tr>
<tr>
<td>Prologue</td>
<td>22</td>
</tr>
</tbody>
</table>
Interfaces ................................................................. 22
External Data Dependencies ........................................... 22
Internal Data Dependencies .......................................... 22
Process/Pseudo code .................................................. 22
Application Layer Design ........................................... 23
  General ..................................................................... 23
  App Communication .................................................. 24
    Facilitator Module .................................................. 24
  GUI .......................................................................... 26
    Presentation ............................................................. 26
  App Processing ......................................................... 28
    Processing Module ................................................... 28
    Logger Module ......................................................... 31
  Database .................................................................... 33
    Dbmanager Module ................................................... 33
Movement Layer Design .............................................. 36
  Overview .................................................................... 36
  PPM Signal Generator Subsystem .................................. 36
    RC Vehicle Configurator ............................................. 36
    Movement Command Interpreter ................................. 38
  Emergency Stop Subsystem ......................................... 39
    Emergency Stop ....................................................... 39
Quality Assurance ........................................................ 40
  Test Plans and Procedures .......................................... 40
    General .................................................................... 40
    Module/Unit Test ..................................................... 40
    Component Test ...................................................... 40
    Integration Test ....................................................... 41
    System Verification Test ............................................. 41
Requirements Traceability Matrix ................................... 42
  Requirements Traceability Matrix ................................. 42
  Module Level Traceability Matrix ................................. 43
Acceptance Plan ........................................................ 44
General .................................................................................................................. 44
Packaging and Installation .................................................................................. 44
Criteria and Testing .......................................................................................... 44
Hardware Components ...................................................................................... 45
Overview ........................................................................................................... 45
MaxSonar EZ1 Ultra-Sonic Sensor (4) ............................................................... 45
  Purpose ........................................................................................................... 45
  Specifications ............................................................................................... 45
  Interfaces ...................................................................................................... 45
Arduino Mega ADK R3 Microcontroller ............................................................ 46
  Purpose ........................................................................................................ 46
  Specifications ............................................................................................... 46
  Interfaces ...................................................................................................... 46
Android Phone – Nexus One .............................................................................. 47
  Purpose ........................................................................................................ 47
  Specifications ............................................................................................... 47
  Interfaces ...................................................................................................... 47
XBee Pro Wire Antenna - Series 2 .................................................................... 48
  Purpose ........................................................................................................ 48
  Specifications ............................................................................................... 48
  Interfaces ...................................................................................................... 48
Traxxas E-Maxx RC Vehicle .............................................................................. 49
  Purpose ........................................................................................................ 49
  Specifications ............................................................................................... 49
  Interfaces ...................................................................................................... 49
Appendices ......................................................................................................... 50
Acronyms .......................................................................................................... 50
Definitions ......................................................................................................... 50
## List of Figures

<table>
<thead>
<tr>
<th>Figure #</th>
<th>Title</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Layer Overview</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Layers with Subsystem</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Sensory Layer Design</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Navigation Layer Decomposition</td>
<td>16</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Application Layer Modules</td>
<td>23</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Facilitator</td>
<td>24</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Presentation</td>
<td>26</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Processing Module</td>
<td>28</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Logger Module</td>
<td>31</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Movement Layer</td>
<td>36</td>
</tr>
<tr>
<td>Table #</td>
<td>Title</td>
<td>Page #</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Table 1</td>
<td>Data Flows</td>
<td>5</td>
</tr>
<tr>
<td>Table 2</td>
<td>Consumer Subsystem Tables</td>
<td>7</td>
</tr>
<tr>
<td>Table 3</td>
<td>Layer Level Producer-Consumer Matrix</td>
<td>9</td>
</tr>
<tr>
<td>Table 4</td>
<td>Requirements Traceability Matrix</td>
<td>42</td>
</tr>
<tr>
<td>Table 5</td>
<td>Module Level Traceability Matrix</td>
<td>43</td>
</tr>
</tbody>
</table>
Introduction

Overview

The Detailed Design Specification (DDS) will illustrate a very detailed overview of Project Jacobia. This document will expand upon the basis set in the Architectural Design Document (ADS). This expansion will consist of delving deeper into the logical tasks that each subsystem has been assigned. Through scrutinizing each subsystem’s responsibility, this document will describe each subsystem’s tasks down to their lowest level components.

Purpose

The purpose of the DDS is to define each aforementioned component in regards to their individual responsibility, structure, process of execution, data structures, and the interfaces of that particular component. Furthermore, this document will also strive to provide a detailed design of the hardware and software present in the system, as well as their interfaces.

Project Scope

The module should provide an existing RC vehicle with the capabilities to operate autonomously. The RC vehicle would be of a specific type and size but would work generally with any type. The vehicle should be able to receive GPS coordinates from a user and travel from its current GPS location to the coordinates given. While travelling to the new location, the vehicle should be able to detect local objects and avoid them as necessary. This module could be used for certain purposes such as autonomous vehicle competitions, exploration, or any other autonomous purpose.

There are several customer requirements that comprise the majority of Jacobia and as such, define the scope of this system. These are the requirements that will most likely take the largest amount of time to implement, and as such, Team Autono-Mo will be giving them the most attention. Those requirements are as follows:

- Customer Requirement 3.15: The system shall be able to travel from one GPS coordinate to another GPS coordinate.
- Customer Requirement 3.17: The system shall be able to avoid local obstacles along its path.
- Customer Requirement 3.4: The system shall be able to interface with standard RC vehicle PPM signals.
- Customer Requirement 3.21: The system shall be able to interface wirelessly with a host application.
- Customer Requirement 3.27: The system shall have a GUI for interfacing with the user.

There are several other important factors to note: Team Autono-Mo will not modify the RC vehicle used in any manner, except for connecting the RC servo wires to Jacobia; the RC vehicle will not be allowed to drive over unfavorable terrain or in unfavorable conditions.
Architecture Overview

General

This section of the document will serve as a brief overview in order to provide the backdrop of the architecture of Project Jacobia. This will provide an introduction into the detailed aspects of the design by introducing a higher-level view of the system to ease the transition into a lower-level, component-based design.

Layer-Name and Description

The Jacobia architecture consists of 4 layers as seen below in Figure 1.

![Figure 1 - Layer Overview](image)

**Sensory Layer**

The sensory layer is responsible for acquiring external sensory data. That external sensory data includes visual sensor data, heading data, and GPS data. Once this layer has acquired the aforementioned data, it is then responsible for transmitting this data to the Navigation Layer.
Navigation Layer

The Navigation Layer is responsible for determining and directing the navigation of the RC vehicle that the system is housed on. This layer receives sensory information from the Sensory Layer and uses it to determine the make-up of its immediate surroundings, positioning, and a path towards its final destination. After determining those things, this layer then instructs the Movement Layer to set the RC Vehicle on the determined trajectory. Also, the Navigation Layer then sends information about the RC vehicle’s journey to the Application Layer.

Application Layer

The Application Layer is responsible for facilitating interactions with the user. Specifically, this layer is responsible for displaying all necessary information to the user. It also prompts the user for needed input and passes this input to the Navigation Layer. This layer also houses the configuration files that the Navigation Layer needs in order to configure the system to the specific RC vehicle that the user chooses to use the product on.

Movement Layer

The Movement Layer is the layer that directly interfaces with the RC vehicle. It is this layer’s responsibility to generate the PPM signals necessary to direct the vehicle’s movement. It is also this layer’s responsibility to load configuration files based on the type of RC vehicle the user selects. The final responsibility of this layer is to provide a means of an emergency stop by turning off all power to the servos.
Inter-Subsystem Dataflow

Overview

Furthermore, each layer is comprised of subsystems as shown in Figure 2.

Figure 2 - Layers with Subsystem

Each subsystem has at least one logical role that it must play in the functionality of its parent layer. These logical roles are the basis for the component breakdown that will take place throughout the document.
## Data Flows
The data flows of the system are enumerated in Table 1

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GPS Feedback</td>
<td>Feedback from the GPS location system.</td>
</tr>
<tr>
<td>2. Distance Feedback</td>
<td>Feedback from obstacle location detection.</td>
</tr>
<tr>
<td>3. Heading Feedback</td>
<td>Feedback from heading sensor.</td>
</tr>
<tr>
<td>4. GPS Sensor Signal</td>
<td>The signal from the GPS sensor.</td>
</tr>
<tr>
<td>5. Distance Sensor Signal</td>
<td>The signal from the obstacle distance sensor.</td>
</tr>
<tr>
<td>6. Heading Sensor Signal</td>
<td>The signal from the heading sensor.</td>
</tr>
<tr>
<td>7. Conditioned Location Information</td>
<td>The GPS location information conditioned for the location subsystem.</td>
</tr>
<tr>
<td>8. Conditioned Heading Information</td>
<td>The heading information conditioned for the navigation system.</td>
</tr>
<tr>
<td>9. Conditioned Visual Information</td>
<td>The obstacle information conditioned for the obstacle avoidance subsystem.</td>
</tr>
<tr>
<td>10. Calculated Location Information</td>
<td>The calculated location for the system.</td>
</tr>
<tr>
<td>11. Calculated Obstacle Information</td>
<td>The calculated instance of obstacles.</td>
</tr>
<tr>
<td>12. Vehicle Status Information</td>
<td>The calculated instance of information about the vehicle.</td>
</tr>
<tr>
<td>13. GPS Destination Input</td>
<td>Input from the user about the current GPS destination.</td>
</tr>
<tr>
<td>14. Current Calculated Trajectory Information</td>
<td>The calculated trajectory that the vehicle should currently follow.</td>
</tr>
<tr>
<td>15. Sensor Data Request</td>
<td>A request for ultrasonic sensor data.</td>
</tr>
<tr>
<td>16. Sensor Trigger</td>
<td>A trigger to enable each ultrasonic sensor.</td>
</tr>
<tr>
<td>17. Created PPM Signal</td>
<td>The PPM signal sent to the correct servo.</td>
</tr>
<tr>
<td>18. Vehicle Status Information</td>
<td>The calculated instance information about the vehicle.</td>
</tr>
<tr>
<td>19. GPS Destination Input</td>
<td>GPS input from the user.</td>
</tr>
<tr>
<td>20. GPS Destination Input</td>
<td>GPS input from the user.</td>
</tr>
<tr>
<td>21. Vehicle Status Information</td>
<td>The calculated instance information about the vehicle.</td>
</tr>
<tr>
<td>22. Vehicle Status Information</td>
<td>The calculated instance information about the vehicle.</td>
</tr>
<tr>
<td>23. User Input</td>
<td>Input from the user.</td>
</tr>
<tr>
<td>24. User Input</td>
<td>Input from the user.</td>
</tr>
<tr>
<td>25. Vehicle Status Information</td>
<td>The calculated instance information about the vehicle.</td>
</tr>
<tr>
<td>26. User Input About Configuration File</td>
<td>Information about vehicle type from the user.</td>
</tr>
<tr>
<td>27. Information About Configuration File</td>
<td>Information about vehicle type that the user chose.</td>
</tr>
</tbody>
</table>

Table 1 - Data Flows
### Producer-Consumer Relationships

In order to illustrate this on the grand scale of the project, the following tables show the producer-consumer relationships between the subsystems.

<table>
<thead>
<tr>
<th>Consumer Subsystem</th>
<th>Producer Subsystem</th>
<th>GPS Locator</th>
<th>Visual Sensing</th>
<th>Heading Detection</th>
<th>Signal Conditioner</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS Locator</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Visual Sensing</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heading Detection</td>
<td></td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal Conditioner</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consumer Subsystem</td>
<td>Producer Subsystem</td>
<td>Localization</td>
<td>Path Processing</td>
<td>Obstacle Avoidance</td>
<td>Navigation Communication</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>--------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Signal Conditioner</td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Localization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obstacle Avoidance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPM Signal Generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consumer Subsystem</th>
<th>Producer Subsystem</th>
<th>Navigation Communication</th>
<th>App Communication</th>
<th>App Processing</th>
<th>GUI</th>
<th>Database</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Communication</td>
<td></td>
<td>19</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>App Processing</td>
<td></td>
<td>20</td>
<td>22</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td></td>
<td>23</td>
<td></td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Consumer Subsystem Tables
# Layer-Level Producer-Consumer Relationships

<table>
<thead>
<tr>
<th>Producer</th>
<th>Consumer</th>
<th>getSensorData</th>
<th>sendSensorData</th>
<th>getGPS</th>
<th>getCompass</th>
<th>threatCalculation</th>
<th>localizationCalculation</th>
<th>pathCalculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android GPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultrasonic Sensor Array</td>
<td>Android Compass</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android Compass</td>
<td>getSensorData</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getSensorData</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sendSensorData</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getGPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCompass</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>threatCalculation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consumer</th>
<th>Producer</th>
<th>RCVehicle Configurator</th>
<th>MovementCommandInterpreter</th>
<th>Servos</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovementCommandInterpreter</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Navigation Layer</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consumer</th>
<th>Producer</th>
<th>threatCalculation</th>
<th>pathCalculation</th>
<th>localizationCalculation</th>
<th>xbee wireless communicator</th>
<th>facilitator2</th>
<th>sendSensorData</th>
<th>movementCommandInterpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>threatCalculation</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pathCalculation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>localizationCalculation</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xbee wireless communicator</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>facilitator2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sendSensorData</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getGPS</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCompass</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consumer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Producer</td>
</tr>
<tr>
<td>Facilitator</td>
</tr>
<tr>
<td>Presentation</td>
</tr>
<tr>
<td>Processing</td>
</tr>
<tr>
<td>Logger</td>
</tr>
<tr>
<td>Dbmanager</td>
</tr>
<tr>
<td>Navigation Layer</td>
</tr>
</tbody>
</table>

Table 3- Layer Level Producer-Consumer Matrix
Module Decomposition

Now that the layers and subsystems have been introduced, this section of the DDS shall present the modules that each subsystem contains. These modules are the mechanisms through which the subsystems are able to execute their tasks. Therefore, their importance cannot be over emphasized. With this in mind, this section will include many charts and diagrams in order to fully define and explain the role of each module.

Sensory Layer Design

Overview

The Sensory Layer functions as the "eyes" of the system. It allows Jacobia to understand where it is, where it is going and the location of objects around it. Sensory information is conditioned into a usable format then relayed to the Navigation Layer to be used in path finding and obstacle avoidance algorithms.

There are a total of 4 subsystems divided into 7 modules/components in the Sensory Layer.

![Figure 3 - Sensory Layer Design](image-url)
**Android GPS Sensor**

**Prologue**

This component is the GPS sensor found on the Android phone. It will provide information to the Signal Conditioner Subsystem to allow Jacobia to be aware of its current location.

**Interfaces**

The Android GPS sensor interfaces with the Android application providing updated information.

**External Data Dependencies**

The Android GPS sensor depends on the global positioning system providing information.

**Internal Data Dependencies**

The Android GPS sensor depends on the actionListener process in the Android operating system.

**Process/Pseudocode**

The sensor operates in the background of the Android application.

**Ultrasonic Sensor Array**

**Prologue**

This is an array of 4 ultrasonic sensors used for range finding. They will provide information to the Signal Conditioner Subsystem to allow Jacobia to be aware of local obstacles.

**Interfaces**

The ultrasonic array interfaces with the arduino through analog pins for the arduino to read analog data, and through digital pins to accept trigger commands from the arduino.

**External Data Dependencies**

The ultrasonic array depends on the feedback from an ultrasonic pulse that it bounces off of objects.

**Internal Data Dependencies**

The ultrasonic array depends on its own ability to convert the feedback to an analog voltage.

**Process/Pseudocode**
The ultrasonic array gives analog voltage proportional to obstacle distance.

**Android Compass Sensor**

**Prologue**

This component is the compass sensor found on the Android phone. It will provide information to the Signal Conditioner Subsystem about Jacobia’s current orientation.

**Interfaces**

The Android compass sensor interfaces with the Android application providing updated information.

**External Data Dependencies**

The Android compass sensor depends on the global positioning system providing information.

**Internal Data Dependencies**

The Android compass sensor depends on the actionListener process in the Android operating system and the internal magnetometer.

**Process/Pseudocode**

The sensor operates in the background of the Android application.

**getSensorData**

**Prologue**

This module is used to trigger and read each ultrasonic sensor separately. This function is triggered once every 50ms with a timer triggered interrupt in the arduino. Each time it is called, it reads the previous sensor, sets that sensor’s Rx low, stores the result into a global first-in-first-out array, then triggers the next sensor by setting that sensor’s Rx pin high. A bubble sort is then performed on the array to keep the data in order. By the time this function is called again (50ms) the signal conversion to analog will have been completed, making it possible to read the previously triggered sensor. Continuing to call this function every 50ms provides an updated global array of sensor data for each sensor.

**Interfaces**

This module interfaces with the Ultrasonic Sensor Array to trigger each sensor to be activated individually. This will eliminate the interference between sensors.
getSensorData()
{
    //read previously triggered sensor
    temp = analogRead(analogPin[i]);
    //set previously triggered Rx pin to low
    digitalWrite(digitalPin[i], low);
    //activate the next pin
    if (i == 4)
        digitalWrite(digitalPin[0], high);
    else
        digitalWrite(digitalPin[i + 1], high);
    //shift data in array down, pushing oldest data out
    for (j = 0; j < 4; j++)
    {
        sensor[i][j] = sensor[i][j + 1];
    }
    //put analogRead on top of stack
    sensor[i][4] = temp;
    //sort updated sensor array using bubble sort
    for (x = 0; x < 5; x++)
    {
        for (y = 0; y < 4; y++)
        {
            if (sensor[i][y] > sensor[i][y + 1])
            {
                temp = sensor[i][y];
                sensor[i][y] = sensor[i][y + 1];
                sensor[i][y + 1] = temp;
            }
        }
    }
}

sendSensorData

Prologue

This function checks to see if serial data is available from the android. If there is serial data available from the android, then it must check to see if the data is a sensor request or if it is a
movement command. If it is a sensor request, then this function will arrange the data in the global sensor arrays, take the median value of each one, then decide whether there is a threat at each sensor. This data is then compiled into one byte of data where each sensor has 2 bits dedicated to it in the form 00000000, and every two bits represent one of the sensors. If there is no threat, then the two bits for that sensor are 00, but if there is a threat at that sensor then the two bits are 01. This value is then written to the android through serial communication.

Interfaces
This module interfaces with the Android phone.

External Data Dependencies
N/A

Internal Data Dependencies
This module depends on the data gathered by the getSensorData() module.

Process/Pseudocode

sendSensorData() {
    // declare byte variable
    byte data = b00000000;
    // call the threatDetection
    data = threatDetection(sensor[][])

    // write the byte to the android
    Serial.write(data);
}

getGPS()

Prologue

This function uses actionListeners to monitor changes in the android phone’s GPS sensor. This function will be called every 0.5 seconds and the data will be conditioned by setting global variables for latitude and longitude. These global variables can then be used by Path Processing.

Interfaces
This module interfaces with the Android Application running on the Android phone.

External Data Dependencies
N/A

Internal Data Dependencies
N/A

Process/Pseudocode

// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        // Called when a new location is found by the network location provider.
        makeUseOfNewLocation(location);
    }
    public void onStatusChanged(String provider, int status, Bundle extras) {} 
    public void onProviderEnabled(String provider) {} 
    public void onProviderDisabled(String provider) {} 
};

// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);

getCompass

Prologue

This function uses actionListeners to monitor changes in the android phone’s compass. This function will be called every 0.5 seconds and the data will be conditioned by updating a global variable for orientation. This global variable can then be used by Path Processing.

Interfaces
This module interfaces with the Android Application running on the Android phone.

External Data Dependencies
N/A

Internal Data Dependencies
N/A

Process/Pseudo code

//get an instance of the SensorManager
SensorManager sManager = Context.getSystemService(Context.SENSOR_SERVICE)

//get the magnetometer
Sensor magnetometer = sManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)

//SensorEventListener magListener = new SensorEventListener()
{
    void onSensorChanged(SensorEvent event)
    {
        //update orientation
    }
}

//register the listener
sManager.registerListener(magListener,magnetometer, SensorManager.SENSOR_DELAY_NORMAL);
Navigation Layer Design

The Navigation Layer serves as the processing brain of the system. This layer is tasked with the processing intensive tasks that the system must undertake. Those tasks include determining the threat posed to the vehicle from multiple directions in terms of obstacles, determining the location of the vehicle, navigating the vehicle towards the user-specified destination, as well as generating instructions that will later be used by the Movement Layer in order to move the vehicle along its path.

There are a total of 4 subsystems in this layer which house a total of five modules and 1 component.

![Figure 4 Navigation Layer Decomposition](image)

Localization Calculation

Prologue

This module is tasked with the responsibility of supplying the Path Processing subsystem with GPS data specifying the location of the RC vehicle. The module accomplishes this task by calling the Signal Conditioning’s subsystem’s function “getGPS”. This function will return a GPS object containing two fields: latitude and longitude. The Localization Calculation module will in turn relay this data to the Path Processing subsystem’s path calculation module.

Interfaces

This module interfaces with the getGPS module contained in the Sensor Conditioning subsystem of the Sensory Layer as well as with the Path Calculation module located in the Path Processing subsystem which is housed by the Navigation Layer.
External Data Dependencies

N/A

Internal Data Dependencies

This module depends on Sensory Layer’s Signal Conditioning Subsystem’s getGPS module to return accurate data in a GPS object data type.

Process/Pseudo code
GPS getLoc()
{
    // this method will be externally called by Path Processing.
    GPS gps;
    // call the Signal conditioner function for my GPS.
    gps = getGPS();
    // this is where extra noise elimination code will take place if needed
    Return gps;
}

Instruction Handler

Prologue

This module is responsible for updating the subsystem in regards commands that it will receive from the Application Layer. Some of its code is located on the Arduino and the other code is located on the Android phone.

Interfaces

This module interfaces with the Application Layer in order to receive the GPS destination coordinate. This module also interfaces with the Path Calculation Module by sending the GPS destination coordinate to the module.

External Data Dependencies

This module depends on the USB connection between the Arduino and the Android phone in order to receive data from the Arduino.

Internal Data Dependencies

This module depends on the XBee component to receive data from the Application Layer.

Process/Pseudo code

Void getCommand()
{
    // this code is located on arduino
    // this will get data from the xbee
    byte msg = Receive();
    // sends the data from arduino to the Android phone over the USB connection.
Path Calculation

Prologue

This module is responsible for running the path processing algorithm of the system in order to create a path for the RC vehicle while considering current heading versus desired heading, speed of the vehicle, and potential obstacles. The algorithm used will be a PD controller.

Interfaces

This module interfaces with the Sensory Layer’s Signal Conditioning Subsystem’s getCompass module, the Localization Calculation Module, as well as the Threat Detection Module. From the getCompass Module, it receives the heading information of the RC vehicle. From the Localization Calculation Module, it receives a GPS coordinate of the RC vehicle’s location. From the Threat Detection Module, it receives threat levels in the direction of each sensor. Lastly, it sends movement commands to the Movement Layer’s Movement Command Interpreter Module.
External Data Dependencies

This module depends on the USB connection between the Arduino and the Android phone on which this code is located.

Internal Data Dependencies

This module depends on Sensory Layer’s Signal Conditioning Subsystem’s getCompass Module to return accurate data as well as the Localization Calculation Module and the Threat Detection Module.

Process/Pseudo code

```c
void getPath(){
  GPS currentLocation; //GPS object
  Int heading; //degrees
  Byte threatCode;
  global GPS destinationHeading;
  //make call for location from localization
  currentLocation = getLoc();
  //make call for heading information from signal conditioner
  heading = getCompass();
  //read USB line from arduino to Phone
  threatCode = readUSB();
  //based on threat posed from the conditioners find out if we need to deviate from our course
  int desiredHeading = currentHeading - destinationGPS;
  //use u = Kpe +/- Kde’ algorithm to find the command for navigation we need to send to
  Movement Layer.
  String directionCommand = // this will be determined by the location of obstacles;
  String intensityCommand = // this will be determined by the result of the threatCode from
  obstacle avoidance. I will be able to determine the threat which will translate directly to
  the level of intensity.
  String stopCommand = //this will be determined by whether or not I need to send two movement
  commands in succession.
  byte MCode = M; //this actually saves 01001101. The ascii value of M.
  byte directionCode, intensityCode, stopCode;

  switch(directionCommand)
  {
    Case Turn Right:
      directionCommand = 00000001;
    Case Turn Left:
      directionCommand = 00000010;
    Case Go Straight:
      directionCommand = 0000011;
    Case Go Straight from Stationary:
      directionCommand = 0000100;
    Case Turn Around:
      directionCommand = 0000101;
    Case Go Backwards:
      directionCommand = 0000110;
    Case Stop: // will model braking by varying level to go straight
      directionCommand = 0000111;
  }
  Switch(intensityCommand)
  {
    Case 1:
      intensityCode = 001000;
    case 2:
```
```
intensityCode = 00010000;
case 3:
    Intensity = 00011000;
}
Switch(stopCommand)
{
    Case (no)
    stopCode = 00000000; //more Movement instructions to come
    Case (yes)
    stopCode = 11000000; // no more movement instructions to come
    // this would occur if we need to send a slowdown then a turn.
}  //combine all these codes into one and send to Movement Layer
code = directionCode + intensityCode + stopCode;
USBwrite(M);
USBWrite(code); //write to USB which is connected to Arduino

**Threat Calculation**

**Prologue**

This module is assigned the task of calculating the threat posed by obstacles around the RC vehicle. The module receives conditioned sensory readings from the Sensory Layer’s Signal Conditioning Subsystem’s getSensorData component by having its “threatDetection” function called by Signal conditioning. The module receives a two dimensional array. The rows signify the sensor number and the column values represent sensor readings of distance to objects. This module only reads the 4th element of the column space because that is the median value. The module then compares that value to threshold variables named “threatLimit, threatLimit2, threatLimit3” to determine if an obstacle is a threat. Essentially, there are three threat levels: not a threat, threat level 1, and threat level 2. It then, using the comparison as a basis, creates a byte code which it then sends to Path Processing’s Path Calculation module.

**Interfaces**

This module interfaces with the sendSensorData Module contained in the Sensor Conditioning subsystem of the Sensory Layer as well as with the Path Calculation module located in the Path Processing Subsystem which is housed by the Navigation Layer.

**External Data Dependencies**

N/A

**Internal Data Dependencies**

This module depends on Sensory Layer’s Signal Conditioning Subsystem’s getSensorData Module to return accurate data in the form of a two dimensional array.

**Process/Pseudo code**

//get an instance of the SensorManager
void threatDetection(sensor [4][5])
{
    int threatLimit, threatLimit2, threatLimit3; //will have to test to find out what this will be.
    //testing sensor 1 if threat level 2(most dangerous threat)
    If(sensor[0][3] < threatLimit3)
        data = data | 00000011;
    //testing if threat level 1(low level threat)
    else If(sensor[0][3] < threatLimit2)
        data = data | 00000010;
    //testing if no threat
    else If(sensor[0][3] > threatLimit)
        data = data | 00000001;
    //onto sensor 2
    //testing if threat level 2(most dangerous threat)
    If(sensor[1][3] < threatLimit3)
        data = data | 00001100;
    //testing if threat level 1(low level threat)
    else If(sensor[1][3] < threatLimit2)
        data = data | 00001000;
    //testing if no threat
    else If(sensor[1][3] > threatLimit)
        data = data | 00000100;
    //onto sensor 3
    //testing if threat level 2(most dangerous threat)
    If(sensor[2][3] < threatLimit3)
        data = data | 00110000;
    //testing if threat level 1(low level threat)
    else If(sensor[2][3] < threatLimit2)
        data = data | 00100000;
    //testing if no threat
    else If(sensor[2][3] > threatLimit)
        data = data | 00010000;
    //onto sensor 4
    //testing if threat level 2(most dangerous threat)
    If(sensor[3][3] < threatLimit3)
        data = data | 11000000;
    //testing if threat level 1(low level threat)
    else If(sensor[3][3] < threatLimit2)
        data = data | 10000000;
    //testing if no threat
    else If(sensor[3][3] > threatLimit)
        data = data | 01000000;
    //write the byte to the android
    Serial.write(data);
}

Xbee Wireless Communicator

Prologue

This component is tasked with the responsibility of providing a wireless medium through which the Arduino can communicate with the host application on the user’s computer.

Interfaces
The Xbee component will interface with the Xbee component located in the Application Layer’s Application Communication Subsystem in order to create the wireless medium. It receives information from the Path Calculation Module to transmit to the Application Layer.

**External Data Dependencies**

N/A

**Internal Data Dependencies**

The module depends on the Xbee’s used in both the Navigation Layer and the Application Layer being able to establish communication with one another.

**Process/Pseudo code**

All processes are handled internally by the Xbee component.

**Facilitator 2**

**Prologue**

This module is tasked with the responsibility of facilitating the use of the Xbee in order to provide a wireless means of communication between the RC vehicle and the user’s computer which will be running our host application.

**Interfaces**

The module will interface with the XBee. It will be the wireless technology employed for the use of receiving and transmitting information.

**External Data Dependencies**

N/A

**Internal Data Dependencies**

The module depends on the Xbee’s used in both the Navigation Layer and the Application Layer being able to establish communication with one another.

**Process/Pseudo code**

```c
void Send(Byte message) //will be called by Path Calculation module
{
    Xbee.write(message);
}
Byte Receive()
{
    byte input;
    input = Xbee.read();
    return input;
}
```
Application Layer Design

General

This layer is responsible for facilitating interactions with the user. Specifically, this layer is responsible for displaying all necessary information to the user. It also prompts the user for the needed input and passes this input to the Navigation Layer. This layer also houses the configuration files that the Navigation Layer needs in order to configure the system to the specific RC vehicle that the user chooses to use the product on.

There are five modules within the Application Layer and will be dependent on a multitude of service provided by the windows OS of which this layer will be built on. Additionally, this layer will be built on .NET 4.0 Framework and its APIs as well as utilizing external libraries for various input and outputs.

Figure 5 Application Layer Modules
App Communication

Facilitator Module

Figure 6 Facilitator

Overview

The Facilitator Module is responsible for connecting the Application Layer with the Navigation Layer through a simple, standards-based point-to-point communications. The use of XBee hardware component makes it possible to have point-to-point communication. The use of windows OS requires configuration of XBee adapter to prompt for a new connection. The configuration shall be managed by Tera Term application.

Interfaces

This module interfaces with Navigation Communication Module of Navigation Layer and the Processing Module of the App Processing subsystem within the Application Layer.

External Data Dependencies

A list of commands gathered from the Navigation Communication subsystem. Example, AT, ATID, ATSH/ATSL, ATDH/ATDL, ATWR etc.

Produce
Serial Data converted into byte for readable data type like int32, char etc.

**Data Members**

Port1 declares the serial port where the hardware is attached to the computer.

```csharp
SerialPort port1 = new SerialPort();
```

- Enumeration of incoming data to the module

```csharp
public enum textmsgtype
{
    Incoming
};
```

-Determines whether there is incoming data to the module

-Declares number of byte to be used for sending command between interfaces.

```csharp
byte[] serialData = new byte[];
```

-Converts byte array to hex, ascii String and store it.

```csharp
SerialPort port1 = new SerialPort();
```

**Functions**

-An event handler that is subscribed to the InputEvent event of the Processing Module. It takes the InputControlEventArgs object and uses the specific function to perform the task.

```csharp
private void event(object sender, EventArgs e)
{
    // Scanning network
    // Scanning port
    // Send Command to active network
    // Fetching the information
    // Sending back the command
}
```

-Convert Byte Array To Hex String

```csharp
private string ByteArrayToHexString(byte[] data)
{
    StringBuilder sb = new StringBuilder(data.Length * 3);
    foreach (byte b in data)
    
        sb.Append(Convert.ToString(b, 16).PadLeft(2, '0').PadRight(3, ' '));
```
-Convert Byte Array To Ascii String

```csharp
private string ByteArrayToAsciiString(byte[] data)
{
    StringBuilder sb = new StringBuilder(data.Length * 3);
    foreach (byte b in data)
        sb.Append(Convert.ToChar(b));
    return sb.ToString().ToUpper();
}
```

GUI

Presentation

![Diagram of GUI and Presentation](image)

**Figure 7 Presentation**

**Overview**
The Presentation Module is responsible for presenting screens where there is a driver class that handles the transition between the screens. This module uses windows OS Serial APIs for the standard I/O such as keyboards and mice that are handled by the .NET Framework. The Screens we are concerned are as follows.

i. Welcome Screen Page
ii. Car Select Page
iii. Instruction Page
iv. Warning Page
v. Check Connection Page
vi. Establishing Connection Page
vii. Please Wait Page
viii. In Transit Page

**Interfaces**
This module interfaces with Windows OS APIs for standard I/O using .NET Framework and the Processing Module of the App Processing subsystem within the Application Layer.

**External Data Dependencies**

Input from the user as an event such as btn_Click, mouse_clicked, txt_field etc.

**Produces**

Different pages related to the event triggered by the user to the output device, and data collected from the GUI to the processing module.

**Data Members**

-This is a System.Windows.Forms object that will display the pictures.

  ```csharp
  private PictureBox pictureBoxname;
  ```

-This is a System.Windows.Forms object that will display the button and record events related to the button.

  ```csharp
  Private Button buttonname
  ```

-This is a System.Windows.Forms object that will display the combo box and record events related to the use of combo box.

  ```csharp
  private ComboBox comboboxname
  ```

-This is a System.Windows.Forms object that will display the text box and record events related to the use of combo box.

  ```csharp
  private TextBox textboxname
  ```

-This is a System.Windows.Forms object that will display the timer.

  ```csharp
  private Timer timername;
  ```

-This is a System.Windows.Forms object that will display the labels.

  ```csharp
  private Label labelname;
  ```

-This is a System.Windows.Forms object that will display the menu.

  ```csharp
  private ToolStripMenuItem menuname;
  ```
Functions

- Initialize the component for the corresponding screen.

```csharp
private void InitializeComponent()
{
    this.labelname = new System.Windows.Forms.Label();
    this.textboxname = new System.Windows.Forms.TextBox();
    this.listBoxname = new System.Windows.Forms.ListBox();
    this.menuname = new System.Windows.Forms.MenuStrip();
    this.menu = new System.Windows.Forms.ToolStripMenuItem();
    this.buttonname = new System.Windows.Forms.Button();
    this.btnStartCar = new System.Windows.Forms.Button();
    this.btnStopCar = new System.Windows.Forms.Button();
    this.toolStripSeparator1 = new System.Windows.Forms.ToolStripSeparator();
    this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
    this.menuStrip1.SuspendLayout();
}
```

App Processing

Processing Module

![Diagram](image)

Figure 8 Processing Module

Overview

This module is responsible for doing processing for each page that requires processing. There are following classes in GUI subsystem that actually needs processing. They are

i. CarSelectManager
ii. CheckConnectionsManager
iii. TransitManager

**Interfaces**

This module interface with Presentation Module of GUI subsystem in Application layer, Facilitator Module within same layer, Logger Module within the same subsystem and Dbmanager Module of Database subsystem within the Application Layer.

**External Data Dependencies**

-It consumes the user input and the events from the Presentation Module.

**Produces**

-Boolean value of connectivity within Application and the vehicle

**Data Members**

Enumerated Data structure for holing the command for vehicle.

```csharp
Public enum VehicleCommand : byte
{
    Speed_Up,
    Speed_Down,
    Turn_Left,
    Turn_Right,
    Backward,
    Forward
};
```

Variables for vehicle Information

```csharp
private Boolean isConnected;
private String carType;
private int longitude;
private int latitude;
```

**Functions**

-Loading of the GUI

```csharp
private void CarSelectGUI_Load(object sender, EventArgs e)
```
Selection of the Vehicle Type

```csharp
private void comVehicleType_SelectedIndexChanged(object sender, EventArgs e)
{
    CarSelectController vehicle = new CarSelectController();
    if (comVehicleType.SelectedItem.Equals("Car"))
    {
        vehicle.setCarType("Car");
        this.pictureBox1.Image = global::Jacobia.Properties.Resources.rc_car;
        //codes for loading configuration file
    }
    if (comVehicleType.SelectedItem.Equals("Bike"))
    {
        vehicle.setCarType("Bike");
        this.pictureBox1.Image = global::Jacobia.Properties.Resources.rc_bike;
        //codes for loading configuration file
    }
    if (comVehicleType.SelectedItem.Equals("Truck"))
    {
        vehicle.setCarType("Truck");
        this.pictureBox1.Image = global::Jacobia.Properties.Resources.rc_truck;
        //codes for loading configuration file
    }
    if (comVehicleType.SelectedItem.Equals("Tank"))
    {
        vehicle.setCarType("Tank");
        this.pictureBox1.Image = global::Jacobia.Properties.Resources.rc_tank;
        //codes for loading configuration file
    }
}
```

Load of Config File

```csharp
private void loadConfigFileToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("You chose to load a config file...", "Choice", MessageBoxButtons.OK);
}
```
-Start of the Vehicle

```csharp
private void btnStartCar_Click(object sender, EventArgs e)
{
    lbxTravelInfo.Items.Add("> " + getTime() + " Car Started...");
    lblStatusReadout.Text = "In Transit...";
}
```

-Stop of the Vehicle

```csharp
private void btnStopCar_Click(object sender, EventArgs e)
{
    lbxTravelInfo.Items.Add("> " + getTime() + " Car Stopped...");
    lblStatusReadout.Text = "Car stopped...";
}
```

**Logger Module**

![Figure 9 Logger Module](image)

**Overview**

This module is used to log the events, errors and causes of the errors.

**Interfaces**

This module interfaces with the Processing Module within the App Processing subsystem and the Dbmanger module of the Database Subsystem within the Application Layer.

**External Data Dependencies**

- LoggerExceptions for different events
- Unexpected try and catch errors

- Warnings and Message from the event source raised during compilation and runtime of the program.

**Produces**

- Text file with all the logs dated with timestamp.

**Data Members**

```csharp
private StreamWriter streamWriter;
private int indent;
```

**Functions**

**Errors**

```csharp
void eventSource_ErrorRaised(object sender, BuildErrorEventArgs e)
{
    // BuildErrorEventArgs adds LineNumber, ColumnNumber, File, amongst other parameters
    string line = String.Format(": ERROR {{0}}({1},{2}): ", e.File, e.LineNumber, e.ColumnNumber);
    WriteLineWithSenderAndMessage(line, e);
}
```

**Warnings**

```csharp
void eventSource_WarningRaised(object sender, BuildWarningEventArgs e)
{
    // BuildWarningEventArgs adds LineNumber, ColumnNumber, File, amongst other parameters
    string line = String.Format(": Warning {{0}}({1},{2}): ", e.File, e.LineNumber, e.ColumnNumber);
    WriteLineWithSenderAndMessage(line, e);
}
```

- Write a line to the log, adding the SenderName and Message (these parameters are on all MSBuild event argument objects)

```csharp
private void WriteLineWithSenderAndMessage(string line, BuildEventArgs e)
{
    if (0 == String.Compare(e.SenderName, "MSBuild", true /*ignore case*/))
```
```csharp
    {  
      // Well, if the sender name is MSBuild, let's leave it out for  
      // prettiness
      WriteLine(line, e);
    }  
    else  
    {  
      WriteLine(e.SenderName + " : " + line, e);
    }
  }

-Just write a line to the log

```csharp
private void WriteLine(string line, BuildEventArgs e)  
{  
  for (int i = indent; i > 0; i--)  
  {  
    streamWriter.Write(" \t");
  }  
  streamWriter.WriteLine(line + e.Message);
}

**Database**

**Dbmanager Module**

*Overview*
This module is managing the codes for the vehicle type and the log file that are created using the execution of the program. This module helps in debugging as every errors, warnings will be stored in the data base using the .NET Framework.

**Interfaces**
The Dbmanager Module interfaces with Processing Module of App Processing Subsystem and Logger Module of App Processing Subsystem within the Application Layer.

**External Data Dependencies**
-Log files

-Vehicle Codes

**Produces**
-Text file for the logs

-Vehicle Codes
Data Members

private MySqlConnection connection;
private string server;
private string database;
private string uid;
private string password;

Functions

- Open Connection

    private bool OpenConnection()
    {
        try {
            connection.Open();
            return true;
        }
        catch (MySqlException ex) {
            switch (ex.Number) {
                case 0:
                    // connection failure
                    break;
                case 1045:
                    // invalid user
                    break;
            }
            return false;
        }
    }

- Close connection

    private bool CloseConnection()
    {
        try {
            connection.Close();
            return true;
        }
        catch (MySqlException ex) {
            MessageBox.Show(ex.Message);
            return false;
        }
    }
- Insert Statement

```csharp
public void Insert()
{
    string query = "INSERT INTO tableinfo
    //open connection
    if (this.OpenConnection() == true)
    {
        //create command and assign the query and connection from
        the constructor
        MySqlCommand cmd = new MySqlCommand(query, connection);

        //Execute command
        cmd.ExecuteNonQuery();

        //close connection
        this.CloseConnection();
    }
}
```

- Update statement

```csharp
public void Update()
{
    string query = "UPDATE tableinfo
    //Open connection
    if (this.OpenConnection() == true)
    {
        //create mysql command
        MySqlCommand cmd = new MySqlCommand();
        //Assign the query using CommandText
        cmd.CommandText = query;
        //Assign the connection using Connection
        cmd.Connection = connection;
        //Execute query
        cmd.ExecuteNonQuery();
        //close connection
        this.CloseConnection();
    }
}
```
Movement Layer Design

Overview

The Movement Layer acts as the main communicator between the Jacobia module and the servos that control the attached RC vehicle. This layer allows Jacobia to send PPM signals to the servos attached, and move the car in whatever way is necessary to move towards the goal GPS location, as well as to avoid obstacles. The PPM signals that must be generated are completely directed by the inputs from the Navigation Layer, which will tell the Movement Layer how the car should move.

There are only two subsystems in this layer, the PPM Signal Generator Subsystem and the Emergency Stop Subsystem. The PPM Signal Generator Subsystem will contain 2 modules – RC Vehicle Configurator and Movement Command Interpreter. The Emergency Stop Subsystem will contain only 1 module – Emergency Stop.

![Figure 10 Movement Layer](image)

PPM Signal Generator Subsystem

RC Vehicle Configurator

Prologue

The RC Vehicle Configurator is the module of the Movement Layer that allows the correct magnitude of a signal to be sent to the servos on the RC vehicle. This configuration will be handled using predefined configuration files based on what type of RC vehicle Jacobia will be attached to. These configurations will include all amounts of PPM signals needed to turn right, turn left, drive straight, drive backwards, make a u-turn, and to activate the breaks of the vehicle.
**Interfaces**

The RC Vehicle Configurator receives Configuration Details from the Configuration File corresponding to the RC vehicle the user chooses.

**External Data Dependencies**

The RC Vehicle Configurator depends on the Application Layer to communicate which RC vehicle the user has chosen.

**Internal Data Dependencies**

The RC Vehicle Configurator depends on its own code to interpret what bytecodes sent to it mean.

**Process/Pseudocode**

```cpp
void RCVehicleConfigurator()
{
    boolean waitingForSerial = true;
    byte carTypeCode = b00000000;
    pinMode(1, OUTPUT);
    servo1.attach(22); //pin number for motor controller
    servo2.attach(26); //pin number for steering servo
    Serial.begin(9600); //baud rate to send data serially, 9600 by default
    //receive PPM signal lengths from configuration file
    while (waitingForSerial)
    {
        //check for carTypeCode from Xbee/ApplicationLayer
        if (carTypeCode != b00000000)
        {
            waitingForSerial = false;
        }
    }
    switch (carTypeCode)
    {
        case b00000001: //load Traxxis E-maxx
            break;
        case b00000010: //load Traxxis Bandit
            break;
        case b00000011: //load Traxxis Rally VXL
            break;
        case b00000100: //load Traxxis For Mustang VXL
            break;
        case b00000101: //load Traxxis E-Revo VXL
            break;
    }
}
```
Movement Command Interpreter

Module Prologue

The Movement Command Interpreter is the module of the Movement Layer that allows Jacobia to understand what the Movement Commands are that are sent from Navigation Layer, and then create the appropriate PPM signals to move the vehicle.

Interfaces

The Movement Command Interpreter receives Movement Commands from the Navigation Layer. The Movement Command Interpreter also sends PPM Signals to the external servos of the attached RC vehicle.

External Data Dependencies

The Movement Command Interpreter depends on the Navigation Layer to send it Movement Commands based on a pre-defined bytecode.

Internal Data Dependencies

The Movement Command Interpreter depends on its own code to interpret what bytecodes sent to it mean.

Process/Pseudocode

```java
Servo servo1;
Servo servo2;
byte movementCommandCode;
int valueRight, valueLeft, valueStraight, initStraightValue, value180, backVal, breakVal;

void loop()
{
    //receive movementCommandCode from Navigation Layer
    //interpret intensity of command by multiplying by a scalar factor based on the three levels of intensity
    switch(movementCommandCode)
    {
        case 00000001: servo3.write(valueRight);
            break;
        case 00000010: servo3.write(valueLeft);
            break;
        case 00000011: servo3.writeMicroseconds(valueStraight);
            break;
        case 00000100: servo3.writeMicroseconds(initStraightVal);
            break;
    }
}
```
b00000101: servo3.write(value180);
break;
b00000110: servo3.writeMicroseconds(backVal);
break;
b00000111: servo3.writeMicroseconds(breakVal);
break;
}
}

**Emergency Stop Subsystem**

**Emergency Stop**

*Module Prologue*

The Emergency Stop module is the module responsible for stopping the RC vehicle from moving any more if an emergency situation occurs. The Emergency Stop module is activated by just flipping a switch on the Jacobia module and is a last resort for stopping the vehicle.

*Interfaces*

The Emergency Stop only interfaces with the power circuitry of the servo wires in order to remove all power.

*External Data Dependencies*

The Emergency Stop depends on the user flipping the switch on the module.

*Internal Data Dependencies*

The Emergency Stop depends on the circuitry within the module to stop power.

*Process/Pseudocode*

The process of the Emergency Stop is a very simple flip of a switch attached to the module. This switch will stop the power supply to the servos of the RC vehicle.
Quality Assurance

Test Plans and Procedures

General
Test plans and Procedures are the essence of any project to ensure the highest possible quality of the product. The Team Autono-Mo shall perform all the quality test based on the different level. The level for testing are the unit level, which consist of module testing. Then the components will be tested as the subsystem. Another level is the integrating testing, which shall test the interfaces between the subsystem and the Layers. The system testing shall be done for the whole system.

Module/Unit Test

I/O interface testing
Verifying the input provided by human interface devices such as keyboard and mouse give the desired output.

GUI testing
- Testing all the GUI Buttons for Event handler and desired output.
- Testing GUI textboxes, combo boxes for handling text and desired output.

Point to Point Communication testing
- Testing Serial Communication between XBee Devices
- Testing Protocol for the XBee Device and the .NET Framework
- Testing Port Communication with XBee Device.
- Testing Communication with XBee and Arduino

Android Arduino Communication testing
- Testing serial communication between Android and Arduino

Motor Controller testing
- Checking the speed of two different motors for forward and backward.
- Testing of turn on wheel at various angles from 0 to 180 degree

Sensor data acquisition testing
- Verify the acquired data from the individual ultrasonic sensors compared to the actual distance between the sensor and obstacle.

Component Test
- Sensory Layer Performance Testing
  - Shall be able to detect obstacles using all the ultrasonic device.
- Shall be able to get the GPS co-ordinate of current state and periodically update the co-ordinate.
- Shall be able to get the Heading Degrees of the current state and periodically update the direction.
- Shall be able to condition the signal from all the sensory device to real life value.
- Shall be able to communicate with the Android interface for generating the GPS coordinates.
- Shall be able to communicate with the Android interface for generating the Heading data.

**Navigation Layer Performance Testing**
- Shall be able to avoid the obstacle detected.
- Shall be able to plan the local path during the operation of the vehicle.
- Shall be able to localized in the given state to the real world map.
- Shall be able to Navigate to the point of destination.

**Movement Layer Performance Testing**
- Shall be able to control the motor speed
- Shall be able to control the servos speed for change in direction
- Shall be able to generate desired PPM signals.

**Application Layer Performance Testing**
- Shall be able to communicate with the navigation layer
- Shall be able to exchange data between navigation layer and application layer for displaying the current state of the vehicle
- Shall be able to receive serialized commands from the other subsystem
- Shall be able to display the result on the screen.

**Integration Test**
- The Sensory Layer shall be tested by integrating it with the Navigation Layer.
- The Navigation Layer shall be tested by integrating it with the Application Layer and the Movement Layer.
- The Movement Layer shall be tested by integrating it with the Navigation Layer.
- The Application Layer shall be tested by integrating it with the Navigation Layer.

**System Verification Test**
The whole system shall be tested on the autonomous mode, which means the system shall be allowed to run and put under different test condition. The test condition shall include following criteria.

- The obstacle shall be real world obstacle which shall be placed locally in the path of the vehicle.
- The GPS co-ordinate shall be changed now and then for the system to interact accordingly.
## Requirements Traceability Matrix

### Table 4 Requirements Traceability Matrix

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Sensory Layer</th>
<th>Navigation Layer</th>
<th>Movement Layer</th>
<th>Application Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replacement of RC Vehicle Receiver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface with standard RC car PPM Signals</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>System Scale</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Power</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Obstacle Avoidance</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input GPS Destination</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC Vehicle Trajectory</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>RC Vehicle Trajectory Corrections</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Object Detection Sensors</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path-finding Algorithm</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>System Navigation</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Wireless Communication</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Location Information Transmission</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Local Object Information transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Live Video Feed Transmission</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Location Information Reporting and Display</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Configuration File</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>
## Module Level Traceability Matrix

<table>
<thead>
<tr>
<th>Customer Requirements</th>
<th>Sensory Layer</th>
<th>Navigation Layer</th>
<th>Application Layer</th>
<th>Movement Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replacement of RC Vehicle Receiver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface with standard RC car PPM Signals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Scale</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obstacle Avoidance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Input GPS Destination</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RC Vehicle Trajectory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC Vehicle Trajectory Corrections</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object Detection Sensors</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Path-finding Algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Navigation</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wireless Communication</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Location Information Transmission</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Local Object Information Transmission</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Location Information Reporting and Display</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration File</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Module Level Traceability Matrix
Acceptance Plan

General
Jacobia must adhere to the requirements outlined in the Requirements Traceability Matrix. The outlined requirements are the critical requirements which will outline the acceptance criteria.

Packaging and Installation
Jacobia will be packaged as one assembled module. The module will be incased in a plastic case with connections for connecting to the servos and motor controller. Also included will be a CD which will include needed drivers for the XBee, the Application for the remote computer, and the Application for the Android. Paper instructions will also be provided which will give the customer information for installing software components as well as installing hardware. More detailed instructions will be provided on the included CD.

Criteria and Testing
The Quality Assurance section outlines the testing procedures. Only by passing the outlined tests will Jacobia meet the testing requirements.
Hardware Components

Overview
This section describes all of the hardware components that will be utilized in the Jacobia system. This section will encompass the purpose, specifications, and interface of each of the components listed.

MaxSonar EZ1 Ultra-Sonic Sensor (4)

Purpose
The purpose of the MaxSonar EZ1 Ultra-Sonic sensors is to determine the distance from obstacles surrounding the module and the module itself. The sonars will accomplish this by emitting sound waves that reflect off of an object and return to the respective sensor. The difference between the time the signal was sent and the time the signal returns is used to calculate the distance of the object from the module.

Specifications
The Ultra-Sonic sensors will be utilizing a 5V input and outputs an analog signal in the range of 0-5V depending on how far away an obstacle is. The resolution of this device is 1 inch, resulting in an effective range of 512 inches. Each inch is represented by the output of the analog pin increasing by ~9.8 mV/inch. All data received from the sensor will be filtered to remove any noisy data that is acquired.

Interfaces
The Ultra-Sonic sensors will be interfaced with the Arduino Mega ADK R3 Microcontroller by connecting the Analog0 – Analog3 pins to the four sensors mounted on the module.
**Arduino Mega ADK R3 Microcontroller**

**Purpose**
The purpose of the Arduino Mega ADK R3 Microcontroller is to allow us to communicate between an Android phone, the RC vehicle servos, and the Ultra-Sonic sensors mounted on the module. This microcontroller will control capturing all sensory data and filtering that data, as well as communicating this data to the attached Android phone. The Android phone will then relay navigation commands back to the Arduino and the Arduino will send the calculated PPM signals to the attached RC vehicle servos.

**Specifications**
The Arduino Mega ADK R3 Microcontroller will be utilizing a 5V input from the RC vehicle. The R3 is specifically made for communication with an Android phone via the lower USB port on the microcontroller. The Arduino has 54 digital I/O pins, 16 analog input pins, and is based off of the ATmega2560 architecture. This microcontroller also supports 256 KB of flash memory and has a clock speed of 16 MHz.

**Interfaces**
The Arduino Mega ADK R3 Microcontroller will interface with several devices: the Android phone, the Xbee wireless communication device, the RC vehicle servos, and the Ultra-Sonic sensors.
Android Phone – Nexus One

Purpose
The purpose of the Nexus One is to serve as our main navigation processing unit, as well as our GPS and compass sensors. The phone will be able to use onboard devices to calculate the RC vehicle’s trajectory and coupled with the Ultra-Sonic sensors, make navigation decisions to tell the RC vehicle how to operate.

Specifications
The Nexus One uses the Android 2.1 Firmware, but will most likely be updated to 4.0 to support more features and allow easier application development. The Nexus has 512 MC of RAM, a 3.7” display, a 998 MHz CPU clock speed, and supports micro-USB to allow communication with the Arduino.

Interfaces
The Nexus One will be interfacing with the Arduino via a micro-USB cable, as well as with the on-board GPS and compass sensors.
XBee Pro Wire Antenna - Series 2

Purpose
The purpose of the XBee Pro Wire Antenna – Series 2 embedded module is to provide Jacobia with a means of communicating between the module attached to the RC vehicle and the host application on a user’s computer. This will allow Jacobia to transmit all data about the RC vehicle’s trip, as well as receiving commands about where to go from the user and what type of RC vehicle is being used. One XBee module will be attached to the module, and the other will be attached to the computer with the host application loaded onto it.

Specifications
The XBee Pro supports a 3.3V power supply, a 250kbps data rate, 1 mile range, a wire antenna, 6 10-bit ADC input pins and 8 digital IO pins. These pins, the data rate, and the range allow the host application to communicate effectively with the Jacobia module mounted to the RC vehicle.

Interfaces
The XBee Pro will only be physically interfacing with the Arduino Mega ADK R3 microcontroller. It will be connected to the RX/TX pins of the Arduino to allow communication of data wirelessly to the computer with the host application.
Traxxas E-Maxx RC Vehicle

**Purpose**
The purpose of the Traxxas E-Maxx RC Vehicle is to allow Team Autono-Mo to test the Jacobia module and ensure that it properly works and is able to make an RC Vehicle to become autonomous and avoid obstacles. This RC vehicle will not be modified in any way except for removing the motor servo wires from the current RC transceiver and attaching them to the Arduino Mega ADK R3.

**Specifications**
The E-Maxx includes an 8.4V motor controller, a Titan 550 Motor, a heavy-duty transmission, two servos for controlling the wheels of the vehicle, shocks to allow for safe and smooth travel, as well as being of a 1/10th scale. This RC vehicle is extravagant in the realm of available RC vehicles, and will be more than adequate to support all of the functionality that is required of the attached Jacobia module.

**Interfaces**
The E-Maxx will be only interfacing with the Arduino Mega ADK R3 microcontroller via the servos. These servos will be attached on digital pins 22 and 26 of the Arduino in order to relay PPM signals to drive the RC Vehicle.
Appendices

Acronyms
ADS – Architectural Design Specification
DDS – Detailed Design Specification
SRS – System Requirements Specification
RC – Remote Control
GPS – Global Positioning System
GUI – Graphical User Interface

Definitions
PPM Signals – Pulse Position Modulation

Autonomous - Acting independently or having the freedom to do so

Encoder - An electro-mechanical device that converts the angular position or motion of a shaft or axle to an analog or digital code

Signal Conditioning – Converting signals from Analog to Digital and filtering out any unwanted noise

Path processing – Calculating the path that the RC vehicle must take based on all sensory information

Path planning – Planning out the path that the RC vehicle must take in order to make it to the goal

Servo - A small, cheap, mass-produced actuator used for radio control and small robotics

Dead Reckoning – The process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time, and course

AT-Command for checking if there is connection with Xbee component

ATID- Command for showing the Personal Area Network ID that is currently assigned to the radio.

ATSH/ATSL – Command for showing high and low parts of that serial number respectively.

ATCN- Command for dropping out of the command mode.