Software Requirements Specification

An Application of Motion Planning on a Quadrotor

30 October 2013

Abdullah Ömer YAMAÇ  1885458
Jonard DOÇİ       1786862
Özgür URAL       1819622
Ömer Faruk ÖZARSLAN  1751429

Clover
Intentionally left blank
# Table of Contents

1  Introduction .......................................................................................................................... 5

1.1 Problem Definition ............................................................................................................ 5

1.2 Purpose .............................................................................................................................. 6

1.3 Scope .................................................................................................................................. 6

1.4 Definitions, acronyms, and abbreviations ........................................................................ 6

1.5 References .......................................................................................................................... 7

1.6 Overview ............................................................................................................................. 8

2  Overall description .............................................................................................................. 9

2.1 Product perspective .......................................................................................................... 9

2.1.1 System interfaces .......................................................................................................... 9

2.1.2 User interfaces ............................................................................................................ 9

2.1.3 Hardware interfaces .................................................................................................... 9

2.1.4 Software interfaces ...................................................................................................... 10

2.1.5 Communication interfaces ......................................................................................... 10

2.1.6 Memory ....................................................................................................................... 10

2.1.7 Operations .................................................................................................................. 10

2.1.8 Site adaptation requirements ....................................................................................... 11

2.2 Product functions ............................................................................................................ 11

2.2.1 Simulation Functionalities ......................................................................................... 11

2.2.2 Quadcopter Functionalities ....................................................................................... 12

2.3 Constraints ....................................................................................................................... 13

2.4 Assumptions and dependencies ....................................................................................... 13

3  Specific requirements ........................................................................................................ 14

3.1 Interface Requirements .................................................................................................... 14

3.1.1 Main Menu Screen ..................................................................................................... 14

3.1.2 Simulation Screen with Automatic Mode ................................................................. 14

3.1.3 Simulation Screen with Manual Mode ......................................................................... 15
### 3.1.4 Options Screen

**3.1.5 About Screen**

### 3.2 Functional Requirements

- **3.2.1 Simulation Functionalities**
- **3.2.2 Quadrotor Functionalities**

### 3.3 Non-functional Requirements

- **3.3.1 Performance requirements**
- **3.3.2 Design constraints**
- **3.3.3 Software System Attributes**

### 4 Data Model and Description

### 5 Behavioral Model and Description

- **5.1 Description for software behavior**
- **5.2 State Transition Diagrams**

### 6 Planning

- **6.1 Team Structure**
- **6.2 Estimation (Basic Schedule)**
- **6.3 Process Model**

### 7 Conclusion
1 Introduction

This document is a software requirement specification to our graduation project which involves applying motion planning algorithms on a quadrotor. In this document, we will firstly introduce the purpose and scope of this document. Secondly, we will give an overall description of the project. After the overall description of the project, we will state specific requirements, data models and behavioral models with their respective descriptions. Finally, we will introduce our plan and end the document with a conclusion part.

1.1 Problem Definition

Two centuries ago, there was no machine capable of flying; but in 21th century the numbers of flying vehicles have rocketed. This goes to show just how rapidly the technology concerning aircraft changes. Nowadays, the number of unmanned vehicles which are completely autonomous has increased, and extensive research is being made in order to improve their functionality.\[^{1}\] They are given specific missions, commonly a certain route they must follow. But since the developers of these unmanned systems are unable to predict every element of the environment in which the aircraft will travel, collision with foreign objects, such as other aircraft, bird, or any other flying object is highly probable. In order to enable the aircraft to successfully avoid these collisions, algorithms must be designed. Our main goal in this project is to implement such an algorithm for a quadrotor. Since preventing collisions necessitates aggressive maneuvers, the best aircraft adept enough to perform such feats is the quadrotor.\[^{2}\]

In this project, motion planning algorithms will be applied to a quadrotor.\[^{3}\] A program will be designed in order to simulate behavior of a quadrotor which estimates its own motion and/or motion of other objects from the quadrotor’s moving space. We will create a simulation for this project to apply these ideas and also we will try to achieve with real quadrotor in laboratory platform.

The problems that we would like to solve in this project are not pertinent to a special geographical location, but rather can be considered as problems related to certain entities such as the military, security and defense units and as well as academic bodies involved in such research. These entities are the ones that experience and are affected by the kind of problems we intend to solve in this project. For this reason, many researches in well-known universities are constantly struggling to find solutions as efficient as possible compatible with the current technology. For example the University of Pennsylvania and ETH Zurich have specialized research divisions that only deal with motion planning on quadrotors. A commercially successful example is the AR Drone quadrotor manufactured by the French company Parrot.\[^{4}\] The AR Drone quadrotor can be controlled by an application running on iOS or Android.
1.2 Purpose

The purpose of this document is to give complete description of all functions and features of the product in order to create a basis for interfaces, functionalities and design.

Also this document will be quite helpful for customer and supplier about the software product so that they can agree on the functions and features of the software to be developed.

1.3 Scope

The software product which we will develop is a simulation of a flying quadrotor and applying motion planning algorithms to that quadrotor in the simulation. At the same time we will try to apply the same algorithms on a real quadrotor. We will provide an interface to control quadrotor manually, in addition to having the feature of running the algorithm automatically. Through this algorithm the quadrotor can escape from incoming object, can trace an object and so on. The main aim is to create this algorithm and we will apply this algorithm for a specific scenario. It can be security of buildings or any military purposes.

1.4 Definitions, acronyms, and abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC</td>
<td>Electronic Speed Controller</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electronically Erasable Programmable Read-Only Memory</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static random-access memory</td>
</tr>
<tr>
<td>Arduino</td>
<td>Microprocessor</td>
</tr>
<tr>
<td>XBee</td>
<td>Wireless Serial Communication Hardware</td>
</tr>
<tr>
<td>Arduino-XBee Shield</td>
<td>It is a component which provides plug in XBee on to Arduino</td>
</tr>
<tr>
<td>CW</td>
<td>Clockwise</td>
</tr>
<tr>
<td>CCW</td>
<td>Counter Clockwise</td>
</tr>
<tr>
<td>X-CTU</td>
<td>It is a software which enables to configure Xbees</td>
</tr>
<tr>
<td>RTS</td>
<td>Request to Send</td>
</tr>
<tr>
<td>CTS</td>
<td>Clear to Send</td>
</tr>
<tr>
<td>Pitch</td>
<td>Rotation about y-axis, as seen in Figure 1</td>
</tr>
<tr>
<td>Roll</td>
<td>Rotation about x-axis, as seen in Figure 1</td>
</tr>
<tr>
<td>Yaw</td>
<td>Rotation about z-axis, as seen in Figure 1</td>
</tr>
</tbody>
</table>

Table 1
1.5 References


4. Parrot AR Drone, retrieved from ardrone2.parrot.com


1.6 Overview

In the following sections of this document, we will be giving information about:

- Overview of the project,
- Specific requirements of the project,
- Data model and behavioral model of the project,
- Planning
- Conclusion
2 Overall description

This section of the SRS should describe the general factors that affect the product and its requirements.

This section usually consists of six subsections, as follows:

a) Product perspective
b) Product functions
c) Constraints
d) Assumptions and dependencies

2.1 Product perspective

The final product of this project will be independent and totally self-contained, as it won’t be a part of a larger system. It will retain its functionalities on its own.

2.1.1 System interfaces

Since our project includes various systems such as the a simulator running in Unity 3d, an application written in Matlab containing the motion planning algorithm and the actual quadrotor with an Arduino Microcontroller attached to it, we will need interfaces that allow the communication between these systems to take place. There will be an interface between the simulation and the Matlab application. This interface will be implemented using sockets. Another interface between the Matlab application and the Arduino microcontroller located on the quadcopter will be needed. This interface will be made possible using Xbee, which is a wireless serial data communicator. The Xbee will allow us to send motion commands to the quadrotor. These interfaces are strictly encapsulated in such a way that the user may be unaware or disinterested in the implementations of such interfaces.

2.1.2 User interfaces

Graphical User Interface (GUI) will be used with the simulation. There will be a user menu and a main simulation scene accessible through menu. Main simulation scene will have 2 modes, which are automatic and manual. We will use Unity 3D for GUI and simulation programming. For further details, see section 3.1.

2.1.3 Hardware interfaces

- Arduino (microprocessor) will communicate with TX/RX port. This port will be connected to Xbee through Arduino Xbee shield.
- Xbee (wireless serial communication device) will provide communication between PC and Arduino. It is connected to PC through USB explorer and also connected to Arduino through
Arduino Xbee Shield. We will create a specific port and configuration to communicate between Xbees.

- ESC (Electronic Speed Controller) will provide to control motors by PWM (Pulse width modulation) signals. Normally it is impossible to control motors with microprocessors. The current flow through microprocessor is very low. Therefore we must feed motors with different source. So we will use this device. The communication between ESC and Arduino will be done through a port.

- Sensors will help for interaction between world and microprocessor. For example to stabilize our quadrotor in hover position.

### 2.1.4 Software interfaces

During the development of our project we will use the following software in order to accomplish our final objective:

- Matlab
- Arduino SDK
- Unity
- X-CTU\(^5\)
- SolidWorks
- Blender

### 2.1.5 Communication interfaces

In our project we have three main systems which are simulation, Matlab and quadrotor. Matlab means that the system will use matlab. Matlab creates some outputs and these outputs will be sent to quadrotor through Xbee which is TX/RX communication hardware. Also matlab sends these outputs to the simulation through a specific socket. Xbee is connected to a PC/laptop by a USB connection. Also another Xbee is connected to quadrotor by Arduino Xbee shield. There is one more communication which is between ESC and Arduino. This communication will be done through a port in the Arduino microcontroller.

### 2.1.6 Memory

Our microprocessor (Arduino Uno) has the following memory properties:

- Flash Memory: 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)

### 2.1.7 Operations

Our project will be operable in two different modes. Two modes will exist for the simulation application and for the quadrotor functionalities respectively. The first mode is the automatic mode. This mode will allow the user of the simulation to specify a target position using the user interface and after having selected the desired target, the user may run the simulation. When the simulation starts running, the quadrotor will automatically go to the target position. The second mode is the
manual mode. This mode will enable the users to directly maneuver the quadrotor in the simulation by using the user interface controls. The user may change the direction and orientation of the quadrotor freely using the aforementioned interface controls. Both these operation modes will also be applicable when controlling the actual quadrotor. The user shall also be able to switch between the two modes of operation freely.

2.1.8 Site adaptation requirements

To run a simulation of a quadcopter and its environment, firstly a model of the quadcopter and the environment shall be built. When trying to change the site of the simulation, the environment model should also be changed and reinitialized.

2.2 Product functions

![Figure 2](image)

2.2.1 Simulation Functionalities

2.2.1.1 Mode Select Functionalities

- Switch to automatic mode
- Switch to manual mode.
2.2.1.2 **Manual Mode Move Functionalities**
- Rise
- Land
- Yaw CW
- Yaw CCW
- Pitch CW
- Pitch CCW
- Roll CW
- Roll CCW.

2.2.1.3 **Automatic Mode Move Functionalities**
- Go to desired location

2.2.1.4 **Simulation Environment Design & Control Functionalities**
- Zoom in
- Zoom out
  - Rotate view CW about x-axis
  - Rotate view CCW about x-axis
  - Rotate view CW about y-axis
  - Rotate view CCW about y-axis
  - Rotate view CW about z-axis
  - Rotate view CCW about z-axis

2.2.2 **Quadcopter Functionalities**

2.2.2.1 **Mode Select Functionalities**
- Switch to automatic mode
- Switch to manual mode.

2.2.2.2 **Manual Mode Move Functionalities**
- Power On
- Power Off
- Hover
- Rise
- Land
- Yaw CW
- Yaw CCW
• Pitch CW
• Pitch CCW
• Roll CW
• Roll CCW.

2.2.2.3  Automatic Mode Move Functionalities
• Go to desired location

2.3  Constraints
There are some constraints which limit our options about quadcopter project.

The first one is the hardware limitations. Our project has signal timing requirements. The signal has to transmit to quadcopter in less than 100ms. Our project will use RTS/CTS (Request to Send / Clear to Send) Signal handshake protocols. Moreover, our Arduino has 32kb flash memory (0.5 KB used by bootloader) limitation, 2 KB SRAM limitation, 1 KB EEPROM limitation and 16 MHz Clock Speed limitation. By the way, one of the most critical limitations is about the battery. The battery has 5100 mAh li-po battery which allows nearly ten minutes fly. In addition, the quadcopter’s motors have weight lift limitations, so adding additional hardware components may constitute a serious problem.

Moreover, our project needs parallel operations. A Matlab program has to do some operations while the Arduino, which is at the quadcopter, also has to do some operations. Both of these operations have to work in a harmony.

Furthermore, our project needs Higher-order languages. We will use C# and JavaScript in our project’s simulation part and we will use Matlab for quadcopter part of the project. We will also use the Arduino programming language when processing the signal data, programming the ESCs and maintaining the stability of the quadrotor.

Lastly, our project has some critical issues. To illustrate some of these issues, if something goes wrong such as in signal transfer, our quadcopter may fall down, lose its balance and cause some damage to itself or its environment.

2.4  Assumptions and dependencies
Our project basically depends on Matlab and Unity. We assumed that these programs are installed in our working environment. Another important dependency is the real quadrotor’s working environment. We will fly the quadrotor in a highly controlled laboratory environment. So our project is decidedly dependent on the quadrotor’s environment.
3 Specific requirements

This section will describe software requirements in detail as subsections which are interface requirements, functional requirements and non-functional requirements of applying motion planning algorithm.

3.1 Interface Requirements

3.1.1 Main Menu Screen

There are five buttons in main screen, namely “Start in Manual Mode”, “Start in Automatic Mode”, “Options”, “About” and “Exit”. Both “Start in Manual Mode” and “Start in Automatic Mode” buttons directs user to the simulation screen with different modes. The “Options” button directs user to the options screen. The “About” button directs user to the about screen. The “Exit” button closes the application.

3.1.2 Simulation Screen with Automatic Mode

In simulation screen with Automatic mode, user will be able to modify start and finish positions at first. After modification user will start or pause simulation with “Space” key until quadrotor reaches final destination. While simulation has started, some information about quadrotor
(coordinates, time elapsed, speed etc.) will be displayed at the top right side. After final destination was reached, stats will be displayed (time elapsed, average speed etc.). User can restart simulation by pressing “Space” key or quit to menu by pressing “Esc” key.

“Esc” key quits to menu screen.

3.1.3 Simulation Screen with Manual Mode

In simulation screen with manual mode, user will be able to modify start position. After pressing “Space” key, user will be able to control quadrotor by using numpad buttons on the keyboard. The “A” key rises the quadrotor, while “Z” key falls. At the top right of the screen some statistical data will be displayed. The simulation continues until user quits by pressing “Esc” key. “Esc” key quits to menu screen.

3.1.4 Options Screen

There will be possible options in this screen. Options will be decided in the design phase. There will be two buttons: “OK” and “Cancel”. The “OK” button applies the settings and returns to main menu, while “Cancel” button returns there without applying settings.

3.1.5 About Screen

Information about developers will be placed here. “Back” button will be return to main menu.

3.2 Functional Requirements

This subsection is a description of each major software function, along with data flow and requirements of the function. We have divided the functionalities in two parts, the parts pertinent to the simulation and the parts pertinent to the quadrotor and matlab part. Although some of these functionalities may appear in both these categories, we showed them to make it clearer for the reader of this document.
3.2.1 Simulation Functionalities

3.2.1.1 Mode Select

3.2.1.1.1 Switch to automatic mode

3.2.1.1.1.1 Diagram:

![Diagram](image)

Figure 3

3.2.1.1.2 Description

This functionality allows the user to change the Operation Mode of the simulation to automatic. In this mode, the quadrotor in the simulation is not controlled by the user, but will fly in auto pilot mode towards the specified target.

3.2.1.2 Switch to manual mode

3.2.1.2.1 Diagram:

![Diagram](image)

Figure 4

3.2.1.2.2 Description

In this use case, the user changes the Operation Mode to manual. In this mode, the quadrotor in the simulation is controlled by the user. The controls are made possible by the user interface that allows the user to precisely control the virtual quadrotor.
3.2.1.2 Manual Mode Move Functionalities

3.2.1.2.1 Rise

3.2.1.2.1.1 Diagram:

![Rise Diagram](image)

3.2.1.2.1.2 Description
In this use case, the user, through the simulator interface, makes the quadrotor take off the ground, or increase its altitude.

3.2.1.2.2 Land

3.2.1.2.2.1 Diagram:

![Land Diagram](image)

3.2.1.2.2.2 Description
In this use case, the user, through the simulator interface, makes the quadrotor land on the ground, or decrease its altitude.

3.2.1.2.3 Yaw CW

3.2.1.2.3.1 Diagram:

![Yaw CW Diagram](image)
3.2.1.2.3.2  Description
In this use case, the user, through the simulator interface, makes the quadrotor yaw in a clockwise direction.

3.2.1.2.4  Yaw CCW

3.2.1.2.4.1  Diagram:

![Diagram for Yaw CCW]

Figure 8

3.2.1.2.4.2  Description
In this use case, the user, through the simulator interface, makes the quadrotor yaw in a counter-clockwise direction.

3.2.1.2.5  Pitch CW

3.2.1.2.5.1  Diagram:

![Diagram for Pitch CW]

Figure 9

3.2.1.2.5.2  Description
In this use case, the user, through the simulator interface, makes the quadrotor pitch in a clockwise direction.
3.2.1.2.6  Pitch CCW

3.2.1.2.6.1  Diagram:

![Diagram of Pitch CCW](image10.png)

3.2.1.2.6.2  Description

In this use case, the user, through the simulator interface, makes the quadrotor pitch in a counter clockwise direction.

3.2.1.2.7  Roll CW

3.2.1.2.7.1  Diagram:

![Diagram of Roll CW](image11.png)

3.2.1.2.7.2  Description

In this use case, the user, through the simulator interface, makes the quadrotor roll in a clockwise direction.

3.2.1.2.8  Roll CCW

3.2.1.2.8.1  Diagram:

![Diagram of Roll CCW](image12.png)
3.2.1.2.8  

**Description**  
In this use case, the user, through the simulator interface, makes the quadrotor roll in a counter-clockwise direction.

---

3.2.1.3  **Automatic Mode Move Functionalities**

3.2.1.3.1  Go to desired location

3.2.1.3.1.1  **Diagram:**

![Diagram](image)

**Figure 13**

3.2.1.3.1.2  **Description**  
In this use case, the user, through the simulator interface, after having entered the automatic mode of operation will make the quadrotor in the simulation go to a desired location.

---

3.2.1.4  **Simulation Environment Design & Control Functionalities**

3.2.1.4.1  Zoom in

3.2.1.4.1.1  **Diagram:**

![Diagram](image)

**Figure 14**

3.2.1.4.1.2  **Description**  
In this use case, the user chooses to zoom in the simulation view in order to inspect small details.
3.2.1.4.2  Zoom out

3.2.1.4.2.1  Diagram:

![Diagram of Zoom out](image1)

**Figure 15**

3.2.1.4.2.2  Description
In this use case, the user, through the simulator interface, chooses to zoom out the simulator view to get a larger picture of the simulated environment.

3.2.1.4.3  Rotate view CW about x-axis

3.2.1.4.3.1  Diagram:

![Diagram of Rotate view CW about x-axis](image2)

**Figure 16**

3.2.1.4.3.2  Description
In this use case, the user, through the simulator interface, chooses to rotate the simulator about the x-axis in a clockwise orientation to get a view from a different angle.

3.2.1.4.4  Rotate view CCW about x-axis

3.2.1.4.4.1  Diagram:

![Diagram of Rotate view CCW about x-axis](image3)

**Figure 17**
3.2.1.4.4.2 Description

In this use case, the user, through the simulator interface, chooses to rotate the simulator about the x-axis in a counter clockwise orientation to get a view from a different angle.

3.2.1.4.5 Rotate view CW about y-axis

3.2.1.4.5.1 Diagram:

![Diagram of Rotate view CW about y-axis]

Figure 18

3.2.1.4.5.2 Description

In this use case, the user, through the simulator interface, chooses to rotate the simulator about the y-axis in a clockwise orientation to get a view from a different angle.

3.2.1.4.6 Rotate view CCW about y-axis

3.2.1.4.6.1.1 Diagram:

![Diagram of Rotate view CCW about y-axis]

Figure 19

3.2.1.4.6.2 Description

In this use case, the user, through the simulator interface, chooses to rotate the simulator about the x-axis in a counter clockwise orientation to get a view from a different angle.
3.2.1.4.7  Rotate view CW about z-axis

3.2.1.4.7.1  Diagram:

![Diagram](image)

Figure 20

3.2.1.4.7.2  Description

In this use case, the user, through the simulator interface, chooses to rotate the simulator about the z-axis in a clockwise orientation to get a view from a different angle.

3.2.1.4.8  Rotate view CCW about z-axis

3.2.1.4.8.1  Diagram:

![Diagram](image)

Figure 21

3.2.1.4.8.2  Description

In this use case, the user, through the simulator interface, chooses to rotate the simulator about the z-axis in a counter clockwise orientation to get a view from a different angle.
3.2.2 Quadrotor Functionalities

3.2.2.1 Mode Select

3.2.2.1.1 Switch to automatic mode

3.2.2.1.1.1 Diagram:

![Diagram showing Switch to automatic mode]

**Figure 22**

3.2.2.1.1.2 Description:
This function enables quadrotor to fly automatically. In this mode the quadrotor move automatically according to scenario. The user only gives input which will be the last position of quadrotor.

3.2.2.1.2 Switch to manual mode

3.2.2.1.2.1 Diagram:

![Diagram showing Switch to manual mode]

**Figure 23**

3.2.2.1.2.2 Description:
This function enables quadrotor to fly manually. In this mode the quadrotor is moved by user. The user can move quadrotor to rise, to down and so on...
3.2.2.2 Manual Mode Move Functionalities

3.2.2.2.1 Power On

3.2.2.2.1.1 Diagram:

![Diagram](image1.png)

Figure 24

3.2.2.2.2 Description:

This function enables quadrotor’s power on while user is away from it. This function helps us to power the quadrotor on in a safe way from a distance. To power up the quadrotor we use a Li-Po battery which is dangerous especially when the connection is setup between battery and rotors. Therefore we are working in more safety position with this function.

3.2.2.2.3 Power Off

3.2.2.2.3.1 Diagram:

![Diagram](image2.png)

Figure 25

3.2.2.2.3.2 Description:

This function enables quadrotor’s power off while user is away from it. While the quadrotor is flying, it may be dangerous. Therefore if we notice that we have any possibility of dangerous position, then we call this function to prevent or reduce possibility of any damage.
3.2.2.4 Hover

3.2.2.4.1 Diagram:

![Diagram of Hover](image1)

This function enables to hold quadrotor in hover position. If user puts a camera on the quadrotor, this function enables to get better image or video or this function can be used in different scenarios.

3.2.2.5 Rise

3.2.2.5.1 Diagram:

![Diagram of Rise](image2)

3.2.2.5.2 Description:

This function enables user to increase the vertical position of quadrotor. This function also can be used for taking off quadrotor.

3.2.2.6 Land

3.2.2.6.1 Diagram:

![Diagram of Land](image3)
3.2.2.2.6.2 Description:

This function enables user to decrease the vertical position of quadrotor. This function also can be used for landing quadrotor.

![Diagram of Yaw CW](image.jpg)

3.2.2.2.6.3 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor roll in a clockwise direction.

3.2.2.2.7 Yaw CCW

3.2.2.2.7.1 Diagram:

![Diagram of Yaw CCW](image.jpg)

3.2.2.2.7.2 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor yaw in a counter clockwise direction.

3.2.2.2.8 Pitch CW

3.2.2.2.8.1 Diagram:

![Diagram of Pitch CW](image.jpg)
3.2.2.8.2 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor pitch in a clockwise direction.

3.2.2.9 Pitch CCW

3.2.2.9.1 Diagram:

![Diagram of Pitch CCW](image)

Figure 32

3.2.2.9.2 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor pitch in a counter clockwise direction.

3.2.2.10 Roll CW

3.2.2.10.1 Diagram:

![Diagram of Roll CW](image)

Figure 33

3.2.2.10.2 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor roll in a clockwise direction.
3.2.2.11 Roll CCW

3.2.2.11.1 Diagram:

![Diagram of Roll CCW](image)

Figure 34

3.2.2.11.2 Description

In this use case, the user, through a matlab interface, sends a command through the Xbee communicator, makes the quadrotor roll in a counter clockwise direction.

3.2.2.3 Automatic Mode Move Functionalities

3.2.2.3.1 Go to desired location

3.2.2.3.1.1 Diagram:

![Diagram of Go to desired location](image)

Figure 35

3.2.2.3.1.2 Description

In this use case, the user will command the quadrotor to go to the location specified. This location is specified in the form of an xyz coordinate by the matlab application and is sent to the quadrotor through the Xbee wireless communicator.

3.3 Non-functional Requirements

3.3.1 Performance requirements

Performance is vital in projects that require real time computations and that contain hardware components. This statement is also true for our project. Since our project is subdivided in 3 parts which are interrelated and will work simultaneously, a high level of performance is expected from the 3 parts, especially the Matlab application which will send commands to the Arduino microcontroller and to the simulation. This is the reason that we chose Matlab as our platform for
implementing motion planning algorithms, since Matlab applications are high performance especially in making calculations of big data, such as three dimensional matrices.

A critical aspect of our project is the algorithm that will allow the quadcopter to hover in space without losing its balance. This algorithm will run on the Arduino microcontroller and it will be processing data received from the sensors located on the quadcopter and fetching the rotation speed of the motors through the ESCs. Thus, the performance of this algorithm is quite essential, since a small delay may cause the quadcopter to quickly lose its balance, fall over and may result in material damage as well as put the nearby human lives in danger.

3.3.2 Design constraints

There are several design constraints to our project. These constraints result from the nature of our project, which embodies different science and engineering disciplines, such as the physical science, electrical and electronic engineering, computer engineering and mechanical engineering. In this subsection we will review such constraints that may limit the functionalities of our final product.

- When trying to control the quadrotor, we should strictly calculate the center of mass of the quadrotor. So when adding new parts on the quadrotor or changing a component we should recalculate the center of mass.
- The Arduino has a limited memory of 32 KB, so we should be careful when processing the sensor data so that we don’t fill all the memory up.
- When applying motion planning algorithms to the quadrotor, the quadrotor’s environment should be known. This limits our ability to test the quadrotor in unknown environments.
- We used a li-po battery to power up our quadrotor. This battery lasts for 15 min, and it requires 3 hours to be fully recharged. This limits our ability to design algorithms that last longer.
- The quadrotor must remain inside a perimeter with radius 30 m from the laptop/computer that runs the matlab application that is connected to the quadrotor. The reason is that the Xbee wireless transmitters have a range of 30 m.
- Unity 3D will be used to run the simulation
- Matlab will be used to run motion planning algorithms and send commands to the Arduino microcontroller and to the simulation
- Arduino SDK will be used to program the Arduino UNO microcontroller
3.3.3 Software System Attributes

3.3.3.1 Reliability

Our system is mostly depending on hardware. We are doing sensor data processing and communications between different hardware components. As a result the reliability of our project highly depends on the reliability of the singular hardware components and their interfaces.

These hardware reliability requirements are:

- Sensor data has to be true at least 99.9% of the time
- Communication must get from the sensor at most 0.1 second.
- Communication between hardware has to be true at least 99% of the time.
- Communication must be done at most 0.3 second.

3.3.3.2 Usability

Our target customers mostly include researchers and security agencies, but we will still create a simple enough interface for ages greater than 13. This interface will comply with design interface standards that make the user's interaction as simple and efficient as possible.

3.3.3.3 Portability

Since we will develop both a simulation and a matlab application that will receive environment data, and send motion commands to a real quadcopter, the portability of our project in general is not well founded. The simulation will be created in Unity 3D, so we will be able to port the simulation as a standalone application in different platforms. But the project as a whole will not be well-portable. Especially since the changing the working environment of the real quadcopter is not feasible, as the working environment of our project as of now is a high-tech specialized laboratory in the MODSIMMER Research Center. This aspect of portability may be later resolved by adding advanced sensors that may take information about the environment.

3.3.3.4 Maintainability

The simulation part will be easily maintainable. Whereas the quadcopter’s maintainability is a little harder to accomplish, since it contains delicate parts and may be subject to damage if comes into contact with other objects. The battery’s life is another concern, since the battery life of a LiPo battery lasts approximately 1 year.

3.3.3.5 Safety

Working with embedded hardware and electrical parts, batteries etc. always contains risks. In addition, the quadcopter’s rotating blades contain life risks, since if they are detached from the rotors, they may fly in unpredicted directions and endanger people’s lives. As a result, we will
conduct extensive testing before finalizing our product so that such risks are decreased at a minimal level. We will also set a minimal age for the users of our product.

### 3.3.3.6 Security

Since our project will not deal with sensitive private data, security is not a real concern. The only matter of interest in the security perspective is the data handled by the Xbee wireless communicators. But since the Xbees encrypt the data that they send/receive, no intruder may interfere with our data.
4 Data Model and Description

In our project we will handle two types of data. The first type is the data received by the different sensors located on the quadrotor. We will store this data for further inspection while testing and developing our project. The other type of data is the data that will be fetched to the Matlab application by a specialized laboratory in the MODSIMMER Research Center. The MODSIMMER Research Center contains some motion detection devices that will detect the motion of the quadrotor. This data contains information about the environment of the quadrotor and the position of the quadrotor and other objects that may reside in the environment. The data will be sent to the Matlab application, which will use the data in the motion planning algorithm. The output of the algorithm will contain commands that will be sent to the quadrotor. We are still researching about the components related to this data.
5 Behavioral Model and Description

5.1 Description for software behavior

The simulation will start in a main menu screen. In the screen, the user will have options for starting the simulation in automatic mode, in manual mode or quit the simulation. The user can pause or resume the simulation. In automatic mode, the user can specify the target in space. Afterwards, the quadrotor will find an efficient path in order to go to the target. There is an interface for the user in order to directly control the motion of the quadrotor in the simulation.

5.2 State Transition Diagrams

In Figure 36 below you can see the state transition diagrams of the states explained in the section above.
6 Planning

6.1 Team Structure

Veysi İşler – Advisor

Serdar Çiftçi – Advisor

Jonard Doçi – Interested Area: Quadrotor, Simulation

Ömer Faruk Özarslan – Interested Area: Simulation, Matlab

Özgür Ural – Interested Area: Quadrotor, Simulation

Abdullah Ömer Yamaç – Interested Area: Quadrotor, Matlab

6.2 Estimation (Basic Schedule)

At the end of first semester we will plan to fly quadrotor and to stable in hover position. At the first semester we have a lot of work which consists of a lot of physics and hardware. In order to gain proper knowledge of the hardware and physical components, we need to conduct some research, which may take some time. We plan to finish at least half of the simulation software before the first semester ends. We will apply the motion planning algorithm during the second semester.

6.3 Process Model

While developing our project we plan to use the agile method of software development. Each week every group member will try to progress in different research directions or developing the software required for our project and every week meetings with the team and our advisor take place, in which we report our progress, evaluate the work done, and set new objectives for the following week. Because our project may be subject to continuous change, we came to the conclusion that this model will be the appropriate model for our graduate project.
7 Conclusion

In this document we have specified the basic requirements for our graduation project, which includes a computer simulation of a quadrotor and the application of motion planning algorithms on the quadrotor itself. Considering the complexity and difficulties in implementing such a large project, we hope to have given clear enough explanations of what is expected from the outcome of this project, all its functionalities and usages.