T.C.
ONDOKUZ MAYIS UNIVERSITY

COMPUTER ENGINEERING

3D MILLING MACHINE DESIGN USING LEGO MİNDSTORMS NXT 2.0® ROBOTIC KIT

UNDERGRADUATE BACHELOR THESIS

Project Students:
Emin Tarık BAKİ
Büşra DEMİR
Funda KARADENİZ
Sefa YILDIZ

Thesis Defence Date : **10 June 2013**

Thesis Consultant **:** Doç. Dr. Erdal KILIÇ

Abstract

In 21th century robotics is one of the most powerful fields in computer science. There are many robotic kits available out of shelf which can be used to create really useful engineering tools and simulate real world machines with a limited cost and time. These kits help youngsters to develop both hardware and software skills in early ages. Students can learn complex engineering tasks easily by developing programmable, smart and autonomous robots.

Lego Mindstorms® robotics kit is one of these robotic kits which allow people to build versatile dynamic machines piece by piece. Its rich programming library and visual integrated programming environment make people design their tools and test them very easily. However, for relatively complex applications Lego's own visual programming environment has a lot of limitations. Compare to main stream programming languages, such as Java and C#, some simple programming tasks such as reading an external file or driving Lego Mindstorms® with any data is very complex. These limitations prevent students implementing more realistic experiments easily. Combination of main stream programming language and Lego's own application programming interface (API) provide a better framework for a deeper level learning experience. Students will gain better insight about robotic and programming concepts, and discover robotic kit's full potential.

The aim of this project is to implement such a framework and demonstrates a 3D milling machine as an example for the framework. The proposed 3D milling machine is built using a Lego Mindstorms NXT 2.0® robotics kit. Two types of programming styles; namely, visual programming and main stream programming, have been implemented and compared for the same milling machine to show that the main stream programming is a better choice while teaching more complex programming and robotic concepts to students.

Keywords

3D Milling Machine, Robotics, Lego Mindstorms NXT 2.0®, Blender, C#, WPF

Index

1. Introduction

1.1 Problem Definition and Motivation

Robotic kits come with their own visual programming environment. Manufacturer of the kits believed that is visual environments help learner to create engineer projects and stimulate students' learning. Their programming tutorials and concentration mostly on their visual environment. There are extensive libraries defining different visual blocks which control different parts of the robotics kit. For simple engineering projects, these blocks one adequate students' needs. It provides a very quick learning of blocks. However, for a relatively complex projects, limitations of visual environment and programming blocks impose severe problems to success of projects. Especially lack of data driven application blocks force students to implement/write motor control/sensor reading tasks in a tedious way. Students have to add extra application blocks for every piece of the project. Simple programming logics such as loop, if else, variable definition becomes very difficult tasks. Nature of quick drag/drop programming interface with visual simulation of robotics parts gives a false sense of achievements to students work require visually exceed is tall drag and drop simplicity of the visual programming environment. A simple loop with few variables is a more common main stream programming language such as JAVA, C-sharp becomes very complex and requires many visual blocks to achieve.

In order to bring simplicity of these programming languages to robotic kits, utilization of Robotic kits low level Application programming interfaces seems a very sensible choice. Lego Mindstorms NXT 2.0 has a well-defined API written in C language as a library. That many wrappers written for this API in order to use other language and frameworks. In practice Lego website lists many implementation of Lego API wrapper's Java, C-sharp, Ruby, JavaScript are among these languages. At the beginning of the projects author's proficiencies were java and c-sharp. Since these two languages are used extensively these both of academically and commercially. There is a big literature, in both languages, which utilize object-oriented nature of these languages. Author's knowledge in c-sharp is better than Java knowledge. For that reason C-sharp is been preferred over Java in this project. The same project could have been implemented in Java using similar object-oriented paradigm and model-view controller (MVC) architecture. C-sharp has been chosen to display if a main stream programming language is used along with the robotic kits API, it will provide a better programming environment over Lego's own visual environment to achieve relatively complex engineering projects.

1.2 Organization of Chapter

In order to create a small CNC machine, three main steps have been defined. First step is finding suitable hardware so that it can be programmed easily for the purpose of this project and the second is finding a 3D designing tool to design 3D objects and, last one is design framework using C# for 3D milling aim.

Second chapter focuses on hardware part of the project and getting basic knowledge of simple software scheme. Electronic prototyping designs like Arduino[3] did, programmable brick and motors from beginning could be created. On the other hand there is an electronic prototyping tool Lego Mindstorms® project is the most suitable source for this purpose due to it includes brick, motors and sensors and software in one bundle has been selected. If we consider today's robot marketing industry and toolkit series. It can be easily programmed and could be built. There are several language library are supported for this toolkit. Furthermore

Lego Mindstorms NXT 2.0® kit has chosen to show that *Lego*® is not just a toy that children could create simple stuff but create more complex tools. An earlier paper Lindh&Holgersson[6] discussed that Effects of Lego® to solve logical problems for pupils'.

Third chapter of this project will describe how 3D objects were designed. An open source 3D object design tool; namely, Blender[4] has been used for designs. Final part of this chapter will include which file system was used to interact with 3D milling machine framework.

Fourth chapter cover details of software part of this project. First part will cover "which software frameworks are works on Lego Mindstorms® brick?" Furthermore there are details of third part of Application AForge.NET[2] which designed in C# language will be explained. Then, the framework which is implemented in this project will be covered in code pieces in details.

Last chapter will focus on important details on this project such as difficulties on this project, which techniques are implemented to overcome these difficulties and conclude "3D Milling Machine with Lego Mindstorms NXT 2.0® using C#" project. Finally this chapter will continue with future ideas and implementations on upper level.

1.3 Restrictions

There are several restrictions on this project.

- The first restriction will be cutting the upper half of convex objects. 3D milling platform will be designed with one drill and, it can only cut upper half of convex objects due to design pattern. We can only cut from top to down for this project. For this reason we could not cut the lower half of convex objects.
- The second restriction is related to cutting mechanism. It is point-by-point cutting mechanism and serve motors speeds are standard. So it will take a considerable amount of time to cut the object fully.
- Last restriction is related to size of the object to be cut. 3D mill's platform is designed for objects which are not bigger than **4.8 cm * 6.8 cm** dimensions. These dimensions based on design implementations of milling machine Y axis weights and reference point of X axis. It could be changeable with different design prototypes.

1.4 Goals and Structure of Project

There are one main goal and there sub-goals in this project.

- Main goal is to build a 3D milling machine using Lego Mindstorms® robotic kit, to program it with right sources and to make it work properly.
- First sub-goal is to understand "*How Lego Mindstorms® works?"* and create some working prototypes using examples provided by Lego Mindstorms®*'s* manual. This prototypes show how to pieces fit together to build more complex structures for the parts of the 3D milling machine. In this stage, the Lego Mindstorms®'s NXT-G software is enough to start with as a starter level programming environment. This sub-aim contains building the 3D machine itself with ordinary Lego and Lego technic pieces.
- The second sub-goal is to design 3D objects in the 3D software tool (Blender) and to export object's coordinate information to a file format so that Lego Mindstorms® brick can interpret and act on. Calibrations of three dimensional coordinates in Blender and their mapping to three axis of 3D milling machine are also part of this sub-goal.
- The third sub-goal is to move further from NXT-G programming language to a more mainstream programming language; namely C# and to use reference libraries of Lego

Mindstorms NXT®'s brick to communicate between the program and the kit. This sub-goal includes rendering 3D objects designed in Blender in the newly written Windows Application and calibration options for different objects.

The last part of the project will conclude the work carried out in this project and will provide some reflections around challenges and difficulties during the project. This part will also give some future directions related to this work.

2. Background
2.1 Lego Mindstorms NXT 2.0 as a Robotics Kits

2.1.1 Introduction to Lego Mindstorms

Following chapter will describe how the Lego Mindstorms 2.0® works as an idea of Lego®. Information about Lego Mindstorms NXT-G® and API libraries which supported by robotic kit will be covered. Last part of this section will cover 3D milling machine as an engineering problem and project methodology details will be explained.

2.1.2 How Lego Mindstorms works?

Lego® is a company which produces sets of little pieces for build some projects for various purposes. From building small cars to big houses, there are lots of options available. Pieces and instructions are in the toolkit set and the aim is building toys according to the instruction manual. The company has firstly started to produce static objects which cannot move then they have improved their products to make more dynamic parts which have gearwheel, motor and sensors for moving objects. Today Lego Mechanics® and Lego Mindstorms® toolkit are robust examples of this purpose. "Lego Mindstorms NXT kits provide an excellent hardware platform for rapid prototyping of sophisticated robotic hardware and makes it very easy to implement examples of the Sense−Plan−Act paradigm with its simple sensor and motor interface"[1] For this project, in order to build the proposed 3D milling machine, one brick with 3 motors for 3 axes x, y, z will be enough. There will also be needed some sensor to control z-axis on one point. Fortunately one toolkit set of Lego Mindstorms NXT 2.0® is covers main project requirements. Meantime, building the cutting platform where 3D objects will be placed requires more solid Lego® pieces that NXT 2.0 ® does not have out of box. Here are Lego Mindstorms® working principles:

1. There is one brick control every motor and sensor.
2. Brick is programmable and you could use either NXT-G programming language for your programs or you can choose libraries in different languages for different APIs.
3. It can be controlled via Universal Serial Bus (USB) cable or wireless Bluetooth technology.

Appendix-A shows main parts of Lego Mindstorms NXT 2.0® which are included in kit and used in this project in details.

What makes Lego robotics such a success?[7]

- First, cheap electronics make small robots affordable.
- Next, Lego bricks have always been a good mechanical prototyping tool.
- Massachusetts Institute of Technology (MIT) researchers have been combining Lego bricks and small computers in interesting ways for years.

2.2 Programming Robotic Kits Visually

The software of NXT is based on National Instruments' LabVIEW[11]. Also, Microsoft developed a platform called Microsoft Robotics Studio (MSRS)[10] which is aimed to support all types of robot. MSRS also provides visual programming environment named Visual Programming Language (MVPL). Both programming environment are in user friendly interface which programmers just have to drag an icon and release it on the main program screen to use it.[9]

2.2.1What is visual programming language Lego® provided for NXT?

LEGO Mindstorms® kit (NXT brick) comes with the NXT-G programming software or LABVIEW for LEGO Mindstorms®. LEGO has released its firmware for the NXT kit as an Open Source project. There are many other unofficial programming languages created to control the brick such as NBC[12] (Next Byte Codes, a simple language with an assembly language syntax), NXC (Not exactly C, a subset of C language built on top of NBC) and leJOS NXJ (2009) (a firmware replacement for Lego Mindstroms' own firmware written in Java language). Microsoft has also released its fourth version of Microsoft Robotic Developer Studio (2012) recently. This programming platform has extensive support for the LEGO brick applications but it has many other not related robotic implementations. This imposes a big learning curve to its users for producing something useful in a relatively short time.

LEGO Mindstroms's® own visual NXT-G software is very useful for prototyping purposes. It helps a lot while trying some small tasks especially in the early phase of prototyping. Source code of basic examples is available from Lego's web site. For that reason, this is a great starting point to understand the concepts and developing initial ideas further related motor and sensor controls. But this software has severe limitations and crashing problems. Since it loads all the code into Lego brick memory at once, it fills the memory very quickly. For complex tasks, such as transformation of 3D object's coordinates into motor rotation values are very cumbersome activity. Wavefront (.obj) file format should be hand coded into the visual language one by one after transforming each vertex coordinates in another tool.

2.3 Programming Robotic Kits API

2.3.1 Core API provided by Robotic kit manufacturer

Following chapter will cover software part of this project. History of LEGO Mindstorms® software will be introduced. Also one section will cover most suitable framework we can use core libraries. *GhostAPI.dll* and, command sets of motors will be covered in details. Last section gives details of framework which is designed for 3D cutting purpose.

During the lifetime of this project, in order to communicate with LEGO Mindstorms® NXT brick, a Windows Application has been written from scratch using available communication libraries provided by LEGO and other third party organizations. In order to provide a framework for developers and hobbyist, LEGO created a Lego Mindstorms® Software development kit (SDK) which contains core "*GhostAPI.dll*" and sample code written in various languages, such as C#. Visual Basic, C++ and J#. Lego has two robotic kits; namely, RCX and NXT kits. RCX was the first robotic kit released by Lego and has limited functionality. NXT kit which this project has employed is LEGO's second generation kit and has more features and advanced capabilities. Lego has created a specific assembly language for these kits called LEGO assembly instruction set (LASM). This instruction set commands can be interpreted and executed with onboard programs or send to the bricks one by one from a PC-driven request. Here are two modes of operation for LEGO brick based applications:

- Mode 1: All the LASM byte code is compiled and downloaded to the LEGO brick as a complete application. In this mode, all the instructions are sent to the brick as a whole program and the brick's processor run these instructions independently. In the early prototyping phase of the project, this mode has been used extensively. When Lego Brick's virtual machine runs a program in this mode, it reads the encoded ".*RXE*" file

format from its flash memory and initializes a 32KB pool of RAM (Random access memory) reserved for use by the user programs. The main functionality of "*.RXE*" is to specify the layout and default content of the 32KB pool. After initialization of this pool, the program is set as active and it is ready to run. While running, the program's main commands are either modifying this pool of RAM or performing Input/output actions based on the values stored in RAM.

- Mode 2: Each LASM byte code is sent to the bricks as a request from a personal computer using either USB port or Bluetooth technology. The main logic of the application stays in the main computer and only single commands are sent to the bricks for processing such as reading values from sensors, changing speed of motors etc. 3D milling machine application uses this mode and controls the brick in an integrated fashion. The NXT firmware uses a master/slave serial port system for Bluetooth communication. Any direct command is interpreted by the NXT Slave firmware and translated into functions. In this mode, the main computer generally sends commands to the NXT brick via Bluetooth packets.

In both modes, a driving personal computer is responsible for assembling and deploying LASM instructions to LEGO bricks. In order to make assembling and deployment easier, LEGO created a set of native Win32 DLLs (dynamic link libraries). The main DLL, "*GhostAPI.dll*" has low level functionalities in order to simply the processing of queuing and deploying the related LASM commands. This DLL uses two more DLLs, namely, "*PbkComm32.dll*" for RS-232 communication and "*PbkUsbPort.dll*" for USB (Universal series bus) communication. In this project, another Microsoft .NET interoperability layer, a third party open source tool, namely "*AForge.NET*" is employed to communicate with the brick. Next section will describe how this communication is performed by the wrapper C# classes and hooking managed code to native code calls. For communication purposes via Bluetooth, some classes provided by Microsoft .NET's own System.dll (System.IO.Ports namespace), such as "SerialPort" class, "StopBits", "Parity" enumerators.

2.3.2 A Third party Wrapper to Core API

AForge.Net[2] framework is a general purpose open source API used for many areas of computer engineering including, robotics, computer vision and artificial intelligence. Whole framework has been written in C# language as independent Visual Studio solutions. All solutions include some sample code related to a specific computer engineering topic and they all use a main core dynamic link library (DLL) called "*AForge.DLL*". This DLL contains core data types and some threading classes. On top of this core DLL, robotic related data types and interfaces have been collected in another DLL called "*AForge.Robotics.Lego.DLL*". This DLL contains two wrapper classes for two types of Lego bricks; namely, NXTBrick and RCXBrick.

"*AForge.Robotics.Lego.DLL*" contains the "*GhostAPI*" wrapper class to LEGO's native Win32 "*GhostAPI.dll*". It uses C#'s "*DllImport*" attribute and "*extern*" keyword to link to corresponding native DLL. It provides hooks for the following "*GhostAPI.dll*" methods:

1. **GhCreateStack:** Creates Ghost communication stack.
2. **GhSelectFirstDevice:** Finds and selects the first available device.
3. **GhOpen:** Opens the currently selected device
4. **GhClose:** Closes the currently selected device.

5. **GhSetInterleave:** Sets the current command "interleave" between executing and download queue.
6. **GhSetWaitMode:** Set the current notification mode to WAIT.
7. **GhCreateCommandQueue:** Creates a command queue (containing one command to start with) and return handle
8. **GhDestroyCommandQueue:** Releases a command queue.
9. **GhAppendCommand:** Appends a command to the given command queue.
10. **GhExecute:** Submits a command queue on the EXECUTE queue.
11. **GhGetFirstCommand:** Gets the first command in the queue.
12. **GhGetCommandReplyLen:** Gets command reply length.
13. **GhGetCommandReply:** Gets command reply.

A typical hooking mechanism in C# is as follows for "*GhCreateStack*". Other methods have been called in the similar fashion.

```
[DllImport ("GhostAPI.dll")]
        public extern static uint uint GhCreateStack(
            string port,
            string protocol,
            string session,
            out IntPtr IntPtr stack );
```

A subset of NXT byte code command is also defined in the "*AForge.Robotics.Lego.DLL*" as C# enumerations in "*NXTCommand*" class. There are three enumerations defines; one for enumeration of command types supported by Lego Mindstorms® NXT brick, one for System commands and one for direct commands.

Command Types:

```
internal enum NXTCommandType
    {
        /// <summary>
        /// Direct command, which requires reply.
        /// </summary>
        DirectCommand = 0x00,

        /// <summary>
        /// System command, which requires reply.
        /// </summary>
        SystemCommand = 0x01,

        /// <summary>
        /// Reply command received from NXT brick.
        /// </summary>
        ReplyCommand = 0x02,

        /// <summary>
        /// Direct command, which does not require reply.
        /// </summary>
        DirectCommandWithoutReply = 0x80,

        /// <summary>
        /// System command, which does not require reply.
        /// </summary>
        SystemCommandWithoutReply = 0x81
    }
```

NXT System Command Types supported by AForge.NET: AForge.Net has implemented a subset of system commands as follows:

```
internal enum NXTSystemCommand
    {
        /// <summary>
```

```
        /// Get firmware version of NXT brick.
        /// </summary>
        GetFirmwareVersion = 0x88,

        /// <summary>
        /// Set NXT brick name.
        /// </summary>
        SetBrickName = 0x98,

        /// <summary>
        /// Get device information.
        /// </summary>
        GetDeviceInfo = 0x9B
    }
```

NXT Direct Command Types supported by AForge.NET: AForge.Net has implemented a subset of direct commands as follows:

```
  internal enum NXTDirectCommand
    {
        /// <summary>
        /// Keep NXT brick alive.
        /// </summary>
        KeepAlive = 0x0D,

        /// <summary>
        /// Play tone of specified frequency.
        /// </summary>
        PlayTone = 0x03,

        /// <summary>
        /// Get battery level.
        /// </summary>
        GetBatteryLevel = 0x0B,

        /// <summary>
        /// Set output state.
        /// </summary>
        SetOutputState = 0x04,

        /// <summary>
        /// Get output state.
        /// </summary>
        GetOutputState = 0x06,

        /// <summary>
        /// Reset motor position.
        /// </summary>
        ResetMotorPosition = 0x0A,

        /// <summary>
        /// Set input mode.
        /// </summary>
        SetInputMode = 0x05,

        /// <summary>
        /// Get input values.
        /// </summary>
        GetInputValues = 0x07,

        /// <summary>
        /// Get status of the Low Speed bus.
        /// </summary>
        LsGetStatus = 0x0E,

        /// <summary>
        /// Write to the Low Speed bus.
        /// </summary>
        LsWrite = 0x0F,

        /// <summary>
        /// Read from the Low Speed bus.
        /// </summary>
```

```
        LsRead = 0x10,

        /// <summary>
        /// Reset input scaled value.
        /// </summary>
        ResetInputScaledValue = 0x08
    }
```

Higher level functions defined in "*NXTBrick*" class uses this low level commands to communicate with the NXT brick. For one high level function, a combination of system and direct commands (with or without reply) has been used in the implementation. In order to send a command to the brick via Bluetooth, the following "*SendCommand*" method in "*NXTBrick*" class is used. "*NXTBrick*" class has a private "*communicationInterface*" member variable which has "*INXTCommunicationInterface*" type and it has low level communication methods such as "*GetMessage*", "*SendMessage*", "*Connect*", "*Disconnect*" etc.

```
public bool SendCommand( byte[] command, byte[] reply )
        {
            bool result = false;

            lock ( sync )
            {
                // check connection
                if ( communicationInterface == null )
                {
                    throw new NullReferenceException( "Not connected to NXT brick" );
                }

                // send message to NXT brick
                if ( communicationInterface.SendMessage( command, command.Length ) )
                {
                    // notifies clients if any
                    if ( MessageSent != null )
                    {
                        MessageSent( this, new CommunicationBufferEventArgs( command ) );
                    }

                    if ( ( command[0] == (byte) NXTCommandType.DirectCommandWithoutReply ) ||
                         ( command[1] == (byte) NXTCommandType.SystemCommandWithoutReply ) )
                    {
                        result = true;
                    }
                    else
                    {
                        int bytesRead;

                        // read message
                        if ( communicationInterface.ReadMessage( reply, out bytesRead ) )
                        {
                            // notifies clients if any
                            if ( MessageRead != null )
                            {
                                MessageRead( this, new CommunicationBufferEventArgs( reply, 0,
bytesRead ) );
                            }

                            // check that reply corresponds to command
                            if ( reply[1] != command[1] )
                                throw new ApplicationException( "Reply does not correspond to
command" );

                            // check for errors
                            if ( reply[2] != 0 )
                            {
                                if ( reply[2] == 221 )
                                {
                                    throw new ApplicationException( "It seems that a wrong
sensor type is connected to the corresponding port" );
                                }
                                else
                                {
```

```
                                    throw new ApplicationException( "Error occurred in NXT
brick. Error code: " + reply[2].ToString( ) );
                                }
                            }

                            result = true;
                        }
                    }
                }
            }

            return result;
        }
```

Many higher level functions are needed to control the LEGO Mindstorm motors and read data from the available NXT sensors. All these methods are defined in the same "*NXTBrick*" wrapper class. For example, in order to connect/disconnect to the brick or control the state of a motor in the robot, following higher level method implementations have been created in aForge.NET.

"Connect" Method:

```
public bool Connect( string portName )
        {
            lock ( sync )
            {
                if ( communicationInterface != null )
                    return true;

                // create communication interface,
                communicationInterface = new SerialCommunication( portName );
                // connect and check if NXT is alive
                if ( ( communicationInterface.Connect( ) ) && ( IsAlive( ) ) )
                    return true;

                Disconnect( );
            }
            return false;
        }
```

"Disconnect" Method:

```
  public void Disconnect( )
        {
            lock ( sync )
            {
                if ( communicationInterface != null )
                {
                    communicationInterface.Disconnect( );
                    communicationInterface = null;
                }
            }
        }
```

"SetMotorState" method:

```
public bool SetMotorState( Motor motor, MotorState state, bool waitReply )
        {
            byte[] command = new byte[12];

            // prepare message
            command[0] = (byte) ( ( waitReply ) ? NXTCommandType.DirectCommand :
NXTCommandType.DirectCommandWithoutReply );
            command[1] = (byte) NXTDirectCommand.SetOutputState;
            command[2] = (byte) motor;
            command[3] = (byte) state.Power;
            command[4] = (byte) state.Mode;
            command[5] = (byte) state.Regulation;
            command[6] = (byte) state.TurnRatio;
```

```
            command[7] = (byte) state.RunState;
            // tacho limit
            command[8]  = (byte) ( state.TachoLimit & 0xFF );
            command[9]  = (byte) ( ( state.TachoLimit >> 8 ) & 0xFF );
            command[10] = (byte) ( ( state.TachoLimit >> 16 ) & 0xFF );
            command[11] = (byte) ( ( state.TachoLimit >> 24 ) & 0xFF );

            return SendCommand( command, new byte[3] );
        }
```

In above code, "*CommunicationBufferEventArgs*" event argument is defined in "*AForge.DLL*", and two exceptions; namely, "*NullReferenceException*" and "*ApplicationException*" are defined in Microsoft.NET "*mscorelib.dll*" system DLL. "*SerialCommunication*" is another class in "*AForge.Robotics.Lego.DLL"*.

3D Milling Machine Application written for this project creates an instance of "*NXTBrick*" class and calls its public methods such as "*Connect*", "*SetMotorState*", "*GetMotorState*" and "*GetSensorValue*" etc. in order to fulfil its functionality.

2.4 Choosing an Engineering: Project 3D milling machine

In many areas of engineering, 3D object designing processes have always been a challenging problem. It desires both software and hardware skills to create a machine which can cut 3D objects created in a 3D software design package. In order to produce any 3D object before CNC (Computer Numerical Control), there were so many manual interventions from drill press to drill holes. Now using CNCs, these tasks have been automated and made mostly error free from human intervention. CNCs are fascinating machines and require many engineering skills to build and operate. For this reason implementing a 3D milling machine with Lego Mindstorms has been classified as a complex engineering project to validate API based programming styles over visual programming.

2.5 Methodology

2.5.1 Project management methodology

An iterative project management methodology has been employed during the life time of this project. Three months have been divided into two weeks iterations. Beginning of each iteration, next two weeks of tasks has been planned. At the end of each iteration, a review of the iteration has been carried out and the next iteration planned accordingly. Each iteration contains both hardware/software implementation and documentation tasks.

2.5.2 Development Methodology and architecture

In this project a test driven programming methodology has been employed. Before writing any code related test cases have been written. Especially, during the calibration phase different test cases has been written to protect 3D milling machine from damaging itself.

.NET Framework provides Model-View-Controller (MVC) implementation that can be used to structure an application. MVC design pattern separate application into three main layers. In these layers different concerns of application; such as presentation, data manipulation and

business logic are implemented independently. This makes architecture of application simpler. In modern applications, this pattern is the main building block which organizes application in a logical structure. Interactions between three layers are shown in Figure 2.6. View layer does not have any knowledge of controller. For that reason more than one controller could be written for a single view. Especially in a test driven development methodology this feature of MVC pattern is important to write independent test cases.

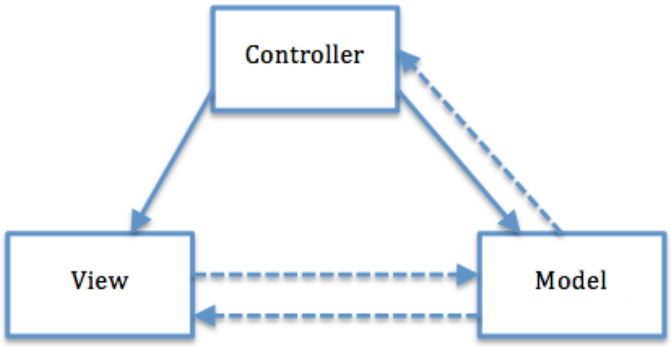


Figure 1: Model-view-controller pattern

3. Solving the Problem

This chapter will cover 3D milling machine platform designing processes and 3D object examples. Furthermore cutting actions which created on NXT-G visual programming interface also will be described and API source codes will be given. End of this chapter will give answer to question "Why NXT-G hadn't been used for all along the project?" Last part will cover 3D cutting concept application interface and test code will be presented.

3.1Designing Hardware for 3D milling machine

3.1.1 Designing prototypes

While building 3D milling machine, many possibilities exists. Especially, possibilities of constructing z-axis and the platform vary remarkably. Some example from the Lego Mindstorms NXT 2.0® manual has been followed and many prototypes are constructed before creating the actual 3D machinery. In the following few sections, different prototypes will be explained and their contribution to building the actual 3D mill will be discussed.

3.1.1.1 First prototype: The Printer

Instructions are taken from **"The Unofficial Lego Mindstorms NXT 2.0 Inventors Guide(2011) "** Chapter: 16 the printer: a drawing machine example has demonstrated. (*Figure 2.1, Figure 2.2*).

Why this design implemented?
This prototype has helped to understand different moving parts of the Lego kit and how they can be built to provide different movements in different directions.

Figure 2.1-Printer prototype front

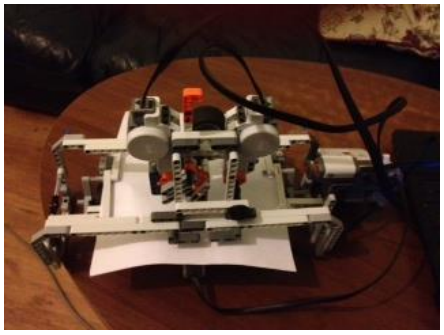


Figure 2.2-Printer prototype top

3.1.1.2 Second prototype: Forklift

Instructions are taken www.nxtprograms.com[13]

Why this design implemented?
It is implemented to give ideas about two axes, which are x-axis and y-axis of 3D milling machine. (*Figure 2.3, Figure 2.4*). This prototype concentrates on accurate control of motors in different axes. A lot of experience has been gained from this prototype in terms of writing small piece of code in Lego Mindstorms NXT 2.0® virtual programming environment.

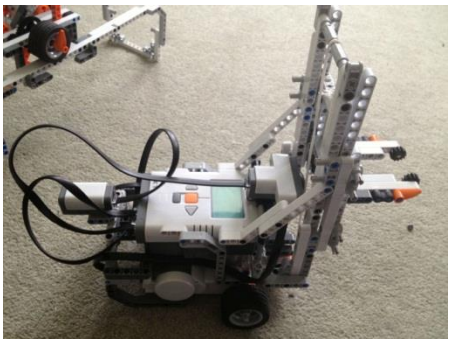


Figure 2.3-Forklift prototype right

Figure 2.4-Forklift prototype top

3.1.1.3 Third prototype: Combined Prototype

Third prototype is unique combination of x, y, z axes. As seen in the following pictures, while z-axis going on the platform, a prototype wheeled vehicle is moving on the directions of x-axis and y-axis.

Why this design implemented?

With this prototype, it was aiming to make it work three axes together, however as it is seen on the picture, this construction wasn't stable for 3D cutting processes. (*Figure 2.5, Figure 2.6*). For that reasons, a more stable platform has been searched.

Figure 2.5-Combined prototype front

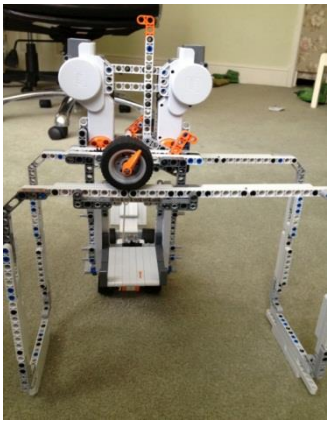


Figure 2.6-Combined prototpye back

3.1.2 3D milling machine prototype

While tree prototypes were built to specific reasons, these implementations help to learn which pieces fit together and how different coordinate system can be constructed in coordination. However, these reference examples didn't help enough to fix the stabilization problem of the platform. For that reason, some other pieces of Lego® products have been employed. After researching various sources, it is found that there are several types of Lego static pieces available for more stable platform.

Furthermore, during the designing process it is realized that the platform needed special racks pieces for stable gearwheel movements for 3 dimensions. Actual 3D milling machine has used these racks extensively for all three dimensions. These racks provide both stability to the 3d mill and more precise control the movement in each dimension.

One of the biggest challenging part of this project is to find missing pieces without know which pieces are missing and built them without any instruction manual. The process of building platform looks like solving a puzzle that you do not know what its picture looks like.

After a consistent study and time management fourth and stable prototype has been completed (*Figure 2.7, Figure 2.8*). **Appendix-B** covers detailed pictures of main part of platform.

Figure 2.7-milling machine(front)

Figure 2.8- milling machine(back)

3.2 Preparing 3D objects for the machine
3D object design on Blender Application

After designing 3D cutting platform there will be 3D object to cut has to be created. After along research Blender application has chosen for this project 3D object design process.

Why Blender?

1. It free to use.
2. There are multiple OS support available. (Windows, Mac Os X, Linux)
3. There are many tutorial and sources are available on the internet.
4. It can export object in Wavefront(.obj) file extension. (Wavefront(.obj) file type is text file which includes coordinates of object and could be recognized by C# language.)

3.2.1. First Design- A pyramid
Why pyramid design?

As a basic level of Blender application it is needed more design details, comparing cube or sphere but if you just look at cutting design of 3D milling machine it is easy to cut edges of Pyramid up to down easily, so Pyramid example is built on blender first.

Design Process:

1) Select two-opposite two points on top of x-axis and slide them on y-axis to make 3 angle prisms. (*Figure 3.1, Figure 3.2*)
2) Then composite points on prism slide on y-axis. (*Figure 3.3*)
3) Here is the final pyramid. (*Figure 3.4, Figure 3.5*)

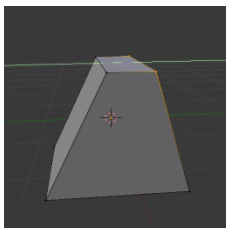


Figure 3.1-Pyramid_1

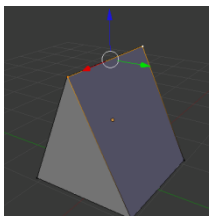


Figure 3.2- Pyramid_2

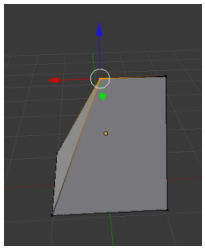


Figure 3.3-Pyramid_3

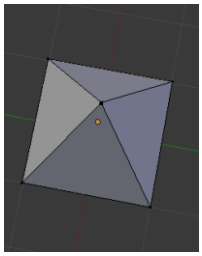


Figure 3.4- Pyramid_4

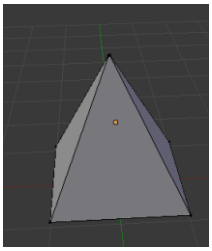


Figure 3.5- Pyramid_5

3.2.2 Second Design-A half sphere

Why half-sphere?
As a fact sphere is easy to implement in Blender but difficult to cut in 3D milling machine. On the other hand it is needed to design half-sphere due to 3D milling machine platform design.
Design Process:

1) Fist UV sphere added on board. (*Figure 3.6*)
2) Second on edit mode every face on lower-half are selected and deleted. (*Figure 3.7, Figure 3.8*)
3) Here is the final half sphere. (*Figure 3.9*)

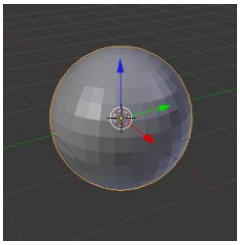


Figure 3.6-Sphere_1

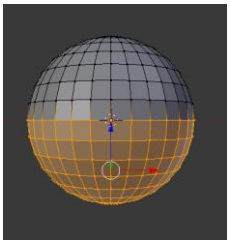


Figure 3.7-Sphere_1

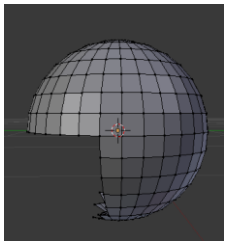


Figure 3.6-Sphere_3

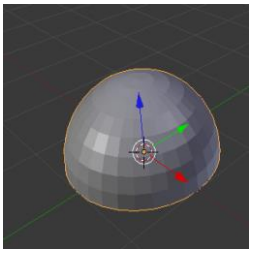


Figure 3.6-Sphere_4

3.2.3 Third Design-A character face

Why Face?
It is difficult to both implement in Blender and, cut in 3D milling machine.
Design Process:

1. Start giving curves to the example cube on the first screen to make it look like a head. Half of the face on z axis is cut and mirror modifier is used for better shape.
2. Eyes are cut in the model. (Figure 3.10, Figure 3.11)
3. Noise is cut in the model.
4. Lips are cut and given shapes to the lips.
5. Shape of head is not smooth so it is made more smooth on the left bar menu. (Figure 3.12, Figure 3.13)
6. Because of the restriction, object is laid down on y-axis and divided half from the center of the y-axis. (Figure 3.14, Figure 3.15, Figure 3.16)

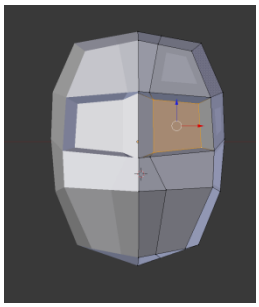


Figure *3.10-Head_1*

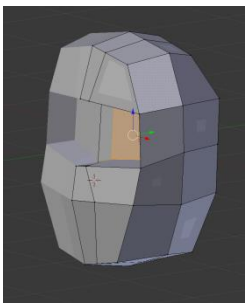


Figure *3.11- Head_2*

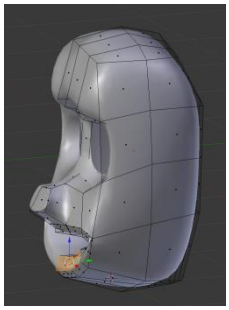


Figure *3.12- Head_3*

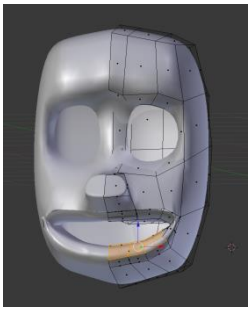


Figure *3.13- Head_4*

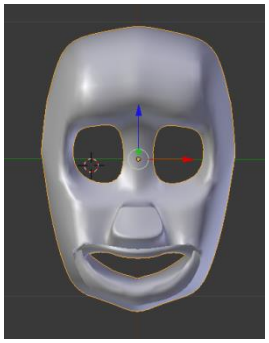


Figure *3.14- Head_5*

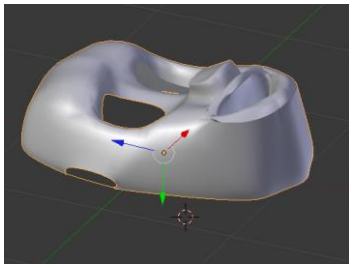


Figure *3.15- Head_6*

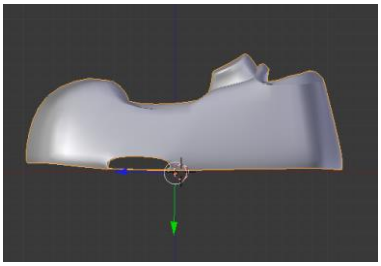


Figure *3.16- Head_7*

3.3 Controlling 3D milling machine visually

3.3.1 Using NXT-G for cutting process

There was one sample programming block which has come with the software itself. This sample module has been used for cutting the objects. In order to apply this block, either this programming block has to be put on the visual programming bench five times, or more complicated repeating logic has to be created. Two more important programming blocks were also needed for keep programming interface simple.

However, NXT-G software was very buggy software and was crashing very easily if it is used for complicated tasks. If more movement in one block has been assigned, the program usually crashes. This is why more stable API was needed for the 3D milling machine. The second important problem while working with NXT-G was its memory limitation. Whenever one program piece is created, it has to be downloaded to the brick before compiled. So that reason memory will become full with complex blocks and it has to be cleaned each time very often.

Details of programming blocks are below:

1) **Cut_process block:** It starts to cut on z-axis and x-axis 5 times. In each cutting process z-axis is decreased one step to make half triangle block. (Figure 3.17) In Figure 3.17 there are some motors and Color sensor are settled in block.
    - First A-motor's details are in *Figure 3.20*.
    - Second A-motor's details are in *Figure 3.21*.
    - B-motor's details are in *Figure 3.22*.
    - Color sensor's details are in *Figure 3.23*.

> **Note ! :** There is one important problem working on NXT-G software. After one block is stopped it waits another parameter to continue and if it continues with another block it causes a problem. So after Cut_process block last function, some motor action should be put end of the block. (Figure 3.18)

2) **Come_0:** It goes one step on y-axis and getting reference starting point on x-axis. (*Figure 3.19*)
    - C-motor details are in *Figure 3.24*
    - B-motor details are in *Figure 2.25*
3) **Simple_ex :** This main program calls **Cut_process** block and **Come_0** block in while loop 5 times.

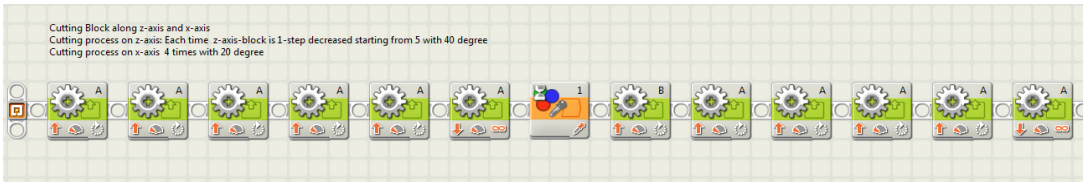


Figure 3.17- Cut process programming block

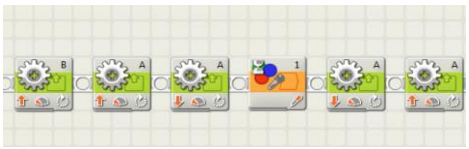


Figure 3.18- Last 2 command are important

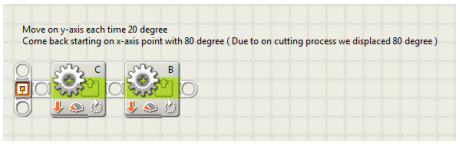


Figure 3.19- Come_0 programming block

Figur 3.20-First A movement

Figure 3.21-Second A movement

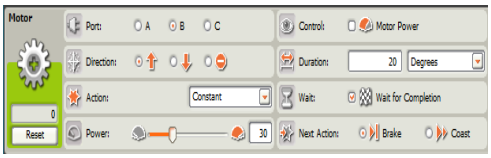


Figure 3.22-B movement

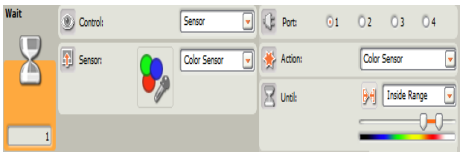


Figure 3.23-Color Sensor

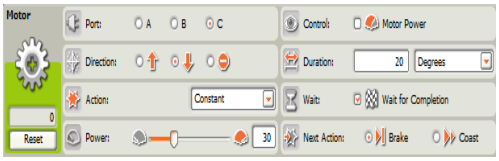


Figure 3.24-Come_0 C movement

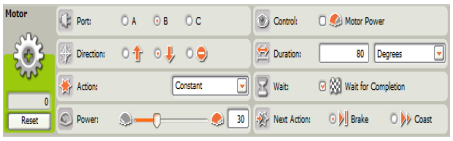


Figure 3.25-Come_0 B movement

Details of motors in pictures are explained below:

1. **Port** : This part show which motor is connected which port of the NXT-Brick
2. **Direction:** It shows direction of motor.
3. **Action:** It indicates how should motor move. Three options are available. Constant, Ramp Up, Ramp Down. This part is available when Duration is **not** unlimited.
4. **Power:** This part can be adjustable to control power of the motor.
5. **Control:** This part is available to control serve motor with motor power.
6. **Duration:** This part is valid to control motor for different acts. There are four options are available. Unlimited, Degrees, Rotations, Second. Default option is Unlimited.
7. **Wait:** If this part is checked it means that motor will wait for Completion of next steps.
8. **Next Action:** This part indicates to act motor according to a selection. If Brake is selected motor will be stopped a moment and if Coast is selected motor will continue to next move without any stoppage time.

3.4. Controlling 3D milling machine using API

3.4.1 3D milling Machine Application

Microsoft WPF (Windows Presentation Foundation) has been used for making the 3D milling machine application. The main Visual Studio solution contains 1 Windows Application and 3 Dynamic Link Libraries (DLL) as shown in *Figure 3.26*.

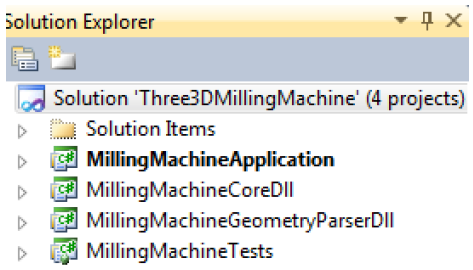


Figure 3.26 – Solution explorer of 3D milling machine project

3.4.1.1 MillingMachineGeometryParserDLL

This dynamic link library has been created to parse "*Wavefront .obj*" file. This file is created by the exporting process in Blender software. The "*Wavefront .obj*" file is a text file and read by the parser created for this project line by line. Parsing operation creates corresponding geometry models based on the information in the file. "*Wavefront .obj*" file has the following structure:

```
# List of Vertices, with (x,y,z[,w]) coordinates, w is optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...

  # Texture coordinates, in (u[,v][,w]) coordinates, v and w are optional and default to 0.
  vt 0.500 -1.352 [0.234]
  vt ...
```

```
...

# Normals in (x,y,z) form; normals might not be unit.
vn 0.707 0.000 0.707
vn ...
...

# Face Definitions
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f ...
...
```

For the purpose of this project, vertex and face information was enough to perform all the functionality required. For face definition, normal vector information has been ignored and "f 1 2 3" format has been adopted. Two main model classes; namely, "*Vertex*" and "*Face*" classes have been created in the related DLL. Instance of these classes are created by parsing "*Wavefront .obj*" file and by selecting the corresponding lines starting with "v" and "f" respectively. How this parsing operation has been carried out is shown in the following "*GeometryParser*" code snippet.

"*GeometryParser*" class: This is the main manager class in this library. Base on the object geometry path, this class reads the corresponding "*Wavefront .obj*" file and creates necessary list containing vertices and faces.

```
    public class GeometryParser
    {

        public List<Vertex> VertexList { get; set; }
        public List<Face> FaceList { get; set; }

        public GeometryParser()
        {
            VertexList = new List<Vertex>(); /// Create new vertex list
            FaceList = new List<Face>(); /// Create new face list
        }

        public void Parse3DGeometry(string folderdir)
        {

            string line;

            double firstV, secondV, thirdV;
            int firstF, secondF, thirdF;
            StreamReader sr = new StreamReader(folderdir); ///  This function is getting
values from file
            while (sr.Peek() > -1) /// Read until file ends
            {
                line = sr.ReadLine();
                if (line[0] == 'v') /// This condition parses the vertices of  the geometry
                {
                    string[] parsedVertices = line.Split('  ');
                    firstV = Convert.ToDouble(parsedVertices[1]);
                    secondV = Convert.ToDouble(parcedVertices[2]);
                    thirdV = Convert.ToDouble(parcedVertices[3]);

                    Vertex myVertex = new Vertex { x = firstV, y = secondV, z = thirdV };
                    VertexList.Add(myVertex); /// Parsed vertices added vertex list

                }
                else if (line[0] == 'f') /// This condition parses the faces of the geometry
                {
                    string[] parsedFaces = line.Split('  ');
                    firstF = Convert.ToInt32(parsedFaces[1]);
                    secondF = Convert.ToInt32(parsedFaces[2]);
                    thirdF = Convert.ToInt32(parsedFaces[3]);

                    Face myFace = new Face { firstIndex = firstF, secondIndex = secondF,
thirdIndex = thirdF };
```

```
                    FaceList.Add(myFace); /// Parsed faces added face list

                }
            }
            sr.Close();
        }
    }
```

"*Vertex*" Class: This class keeps the coordinates of 3D point of the 3D model created in Blender. Coordinates are defined as doubles.

```
    /// <summary>
    /// Vertex: This contains the Vertex coordinates as 3 dimensional Cartesian coordinate
    /// </summary>
    public class Vertex
    {
        public double x { get; set; }
        public double y { get; set; }
        public double z { get; set; }

        public string Serialize()
        {
            return String.Format("Vertex:({0}, {1}, {2})", x, y, z);
        }

        public Vertex Clone()
        {
            return new Vertex { x = this.x, y = this.y, z = this.z};
        }
    }
```

"*Face*" Class: This class keeps the vertex indices of 3D point of the 3D model created in Blender as triangles. Face information has only been used to draw 3D object on the screen. It wasn't use in the cutting operation.

```
    /// <summary>
    /// Face: This class represents the index of vertices for a triangle
    /// </summary>
    public class Face
    {
        public int firstIndex;
        public int secondIndex;
        public int thirdIndex;

        public string Serialize()
        {
            return String.Format("Face:({0}, {1}, {2})", firstIndex, secondIndex, thirdIndex);
        }
    }
```

"*Displacement*" class: In order to simplify calculations of motor rotations, in addition to vertex information, displacement values between two vertices in three dimensions are required. For that reason, a new additional class has been defined in this DLL.

```
    public class Displacement  /// This class is responsible for moving motors one direction
to another
    {
        public double x { get; set; }
        public double y { get; set; }
        public double z { get; set; }
        public Vertex SourceVertex { get; set; }
        public Vertex TargetVertex { get; set; }

        public Displacement()/// Constructor method
        {

        }

        public string Serialize()
        {
            return String.Format("Dsp:({0}, {1}, {2})", x, y, z);
```

```
        }

        public Displacement(Vertex sourceVertex, Vertex targetVertex) /// This method declares
target and source destinations
        {
            x = targetVertex.x - sourceVertex.x;
            y = targetVertex.y - sourceVertex.y;
            z = targetVertex.z - sourceVertex.z;

            SourceVertex = sourceVertex;
            TargetVertex = targetVertex;
        }
    }
```

3.4.1.2 MillingMachineApplication

This application contains three "XAML" files; namely, "*App.xaml*", "*MainWindows.xaml*", and "*GeometryWindow.xaml*". "*App.xaml*" is extended Microsoft's "*Application*" class in "*PresentationFramework.dll*" (System.Windows namespace), which is the main WPF core library came with .NET framework. When the application launches, it creates an instance of "*MainWindows.xaml*" class which contains the program's main user interface, related event handlers and the main "*StartCutting*" method. "*MainWindows.xaml*" class has been extended from "*Window*" class in "*PresentationFramework.dll*" (System.Windows namespace).

All the communication to NXT brick has been carried out by an instance of "*NXTBrick*" class namely "*nxt*" as follows:

```
// NXT brick
        private NXTBrick nxt = new NXTBrick();
```

Different mode of operation has also been defined in this class as follows:

```
// regulation modes
        private NXTBrick.MotorRegulationMode[] regulationModes = new
NXTBrick.MotorRegulationMode[] {
            NXTBrick.MotorRegulationMode.Idle,
            NXTBrick.MotorRegulationMode.Speed,
            NXTBrick.MotorRegulationMode.Sync };
        // run states
        private NXTBrick.MotorRunState[] runStates = new NXTBrick.MotorRunState[] {
            NXTBrick.MotorRunState.Idle,
            NXTBrick.MotorRunState.RampUp,
            NXTBrick.MotorRunState.Running,
            NXTBrick.MotorRunState.RampDown };
        // sensor types
        private NXTBrick.SensorType[] sensorTypes = new NXTBrick.SensorType[] {
            NXTBrick.SensorType.NoSensor, NXTBrick.SensorType.Switch,
            NXTBrick.SensorType.Temperature, NXTBrick.SensorType.Reflection,
            NXTBrick.SensorType.Angle, NXTBrick.SensorType.LightActive,
            NXTBrick.SensorType.LightInactive, NXTBrick.SensorType.SoundDB,
            NXTBrick.SensorType.SoundDBA, NXTBrick.SensorType.Custom,
            NXTBrick.SensorType.Lowspeed, NXTBrick.SensorType.Lowspeed9V,
            NXTBrick.SensorType.Highspeed, NXTBrick.SensorType.ColorFull,
NXTBrick.SensorType.ColorRed,
            NXTBrick.SensorType.ColorGreen, NXTBrick.SensorType.ColorBlue,
NXTBrick.SensorType.ColorNone, NXTBrick.SensorType.ColorExit};
        // sensor modes
        private NXTBrick.SensorMode[] sensorModes = new NXTBrick.SensorMode[] {
            NXTBrick.SensorMode.Raw, NXTBrick.SensorMode.Boolean,
            NXTBrick.SensorMode.TransitionCounter, NXTBrick.SensorMode.PeriodicCounter,
            NXTBrick.SensorMode.PCTFullScale, NXTBrick.SensorMode.Celsius,
            NXTBrick.SensorMode.Fahrenheit, NXTBrick.SensorMode.AngleSteps };
```

In 3D milling machine, three NXT brick motors have been used for running the machine in three dimensions. In "*MainWindows.xaml*" class, these motors are set as follows:

```
        // Returns selected motor
        private NXTBrick.Motor GetSelectedMotorZ()
        {
            return (NXTBrick.Motor)1;
        }

        // Returns selected motor
        private NXTBrick.Motor GetSelectedMotorY()
        {
            return (NXTBrick.Motor)0;
        }

        // Returns selected motor
        private NXTBrick.Motor GetSelectedMotorX()
        {
            return (NXTBrick.Motor)2;
        }

        private NXTBrick.Motor GetThreeDSelectedMotor(int axis)
        {
            return (NXTBrick.Motor)axis;
        }
```

In the same class, another method called "*RunMotorInAnAxis*" method is employed in order to send very high level commands to each NXT motor via "*SetMotorState*" method in AForge.NET's "*NXTBrick*" class as explained in previous section. The body of "*RunMotorInAnAxis*" method is presented as follows:

```
private NXTBrick.MotorState threeDMotorState;
        /// <summary>
        ///
        /// </summary>
        /// <param name="axis"> 0: Y axis (cutting motor), 1: Z axis, 2: X axis</param>
        /// <param name="displacement"></param>
        private bool RunMotorInAnAxis(int axis, int displacement) /// This method includes
basic operations on motors
        {
            if (displacement != 0)
            {
                if (displacement < 0)
                {
                    threeDMotorState.Power = -1 * Convert.ToByte(powerUpDown.Text);
                }
                else
                {
                    threeDMotorState.Power = Convert.ToByte(powerUpDown.Text);
                }

                try
                {
                    threeDMotorState.TachoLimit = Math.Abs(displacement);
                }
                catch
                {
                    threeDMotorState.TachoLimit = 1000;
                    tachoLimitBox.Text = threeDMotorState.TachoLimit.ToString();
                }

                // set motor's state
                if (nxt.SetMotorState(GetThreeDSelectedMotor(axis), threeDMotorState, false)
!= true)
                {
                    System.Diagnostics.Debug.WriteLine("Failed setting motor state");
                }

                return true;
            }

            return false;
        }
```

Actual cutting operation is controlled "*StartCutting*" method in "*MainWindows.xaml*" class. This method brings all the low level definition and implementation of geometry information, motor/sensor control and cutting logic together. The body of this method is presented as follows:

```
/// <summary>
        /// StartCutting: Main Cutting Method. Before calling this method, be sure that the 3d
machine is ready.
        /// </summary>
        /// <param name="calibrationOnly"></param>
        private void StartCutting(bool calibrationOnly)
        {
             MillingMachineManager myMillingMachineManager = new
MillingMachineManager(AppDomain.CurrentDomain.BaseDirectory + @"\head-vise.obj");

             threeDMotorState = new NXTBrick.MotorState();


            // prepare motor's state to set
             threeDMotorState.Power = Convert.ToByte(powerUpDown.Text);
             threeDMotorState.TurnRatio = 0;
             threeDMotorState.Mode = ((modeOnCheck.IsChecked == true) ? NXTBrick.MotorMode.On
: NXTBrick.MotorMode.None) |
                 ((modeBrakeCheck.IsChecked == true) ? NXTBrick.MotorMode.Brake :
NXTBrick.MotorMode.None) |
                 ((modeRegulatedBox.IsChecked == true) ? NXTBrick.MotorMode.Regulated :
NXTBrick.MotorMode.None);
             threeDMotorState.Regulation = regulationModes[regulationModeCombo.SelectedIndex];
             threeDMotorState.RunState = runStates[2];

            // Prepare sensor state
            if (nxt.SetSensorMode(GetSelectedSensor(),
              sensorTypes[BLUE_COLOR_SENSOR_STATE_INDEX],
              sensorModes[0], false) != true)
            {
                System.Diagnostics.Debug.WriteLine("Failed setting input mode");
            }

            List<Displacement> displacementList;

            if (calibrationOnly)
            {
                displacementList =
myMillingMachineManager.GetVertexCalibrationListForMachineCoordinatesAsDisplacement();
            }
            else
            {
                displacementList =
myMillingMachineManager.GetVertexListForMachineCoordinatesAsDisplacement();
            }



            foreach (Displacement v in displacementList)
            {
                // Run X axis motor

                int xDisplacement = Convert.ToInt32(v.x);

                if (RunMotorInAnAxis(2, xDisplacement))
                {
                    Thread.Sleep(Math.Abs(xDisplacement * 100));
                }


                //displayTxt.Text += xDisplacement + ",";

                // Run Z axis motor

                int zDisplacement = Convert.ToInt32(v.z);

                if (RunMotorInAnAxis(1, zDisplacement))
                {
```

```
                    Thread.Sleep(Math.Abs(zDisplacement * 100));
                }


                // This is the main rotating/cutting motor. We use actual vertex information
rather than displacement information and
                // use color detector for returning back

                int yPosition;
                if (calibrationOnly)
                {
                    yPosition = Convert.ToInt32(v.y);
                }
                else
                {
                    yPosition = Convert.ToInt32(v.TargetVertex.y);
                }


                if (RunMotorInAnAxis(0, yPosition))
                {
                    Thread.Sleep(Math.Abs(yPosition * 10));


                    NXTBrick.SensorValues sensorValues;

                    do
                    {
                        // get input values for Sensor
                        if (nxt.GetSensorValue(GetSelectedSensor(), out sensorValues))
                        {
                            validCheck.IsChecked = sensorValues.IsValid;
                            calibratedCheck.IsChecked = sensorValues.IsCalibrated;
                            sensorTypeBox.Text = sensorValues.SensorType.ToString();
                            sensorModeBox.Text = sensorValues.SensorMode.ToString();
                            rawInputBox.Text = sensorValues.Raw.ToString();
                            normalizedInputBox.Text = sensorValues.Normalized.ToString();
                            scaledInputBox.Text = sensorValues.Scaled.ToString();
                            calibratedInputBox.Text = sensorValues.Calibrated.ToString();
                        }
                        else
                        {
                            System.Diagnostics.Debug.WriteLine("Failed getting input values");
                        }


                        // Now go up until blue color is reached
                        threeDMotorState.Power = -1 * Convert.ToByte(powerUpDown.Text);
                        threeDMotorState.TachoLimit = 5;

                        // set motor's state
                        if (nxt.SetMotorState(GetSelectedMotorY(), threeDMotorState, false) !=
true)
                        {
                            System.Diagnostics.Debug.WriteLine("Failed setting motor state");
                        }

                        Thread.Sleep(Math.Abs(50));

                    } while (sensorValues.Raw < 250);
                }
            }
        }
```

"*GeometryWindow.xaml*" class has also been extended from "*Window*" class in "*PresentationFramework.dll*" (System.Windows namespace). This class is responsible for displaying the 3D object in the application.

```
namespace MillingMachineApplication
{
    /// <summary>
    /// Interaction logic for GeometryWindow.xaml
    /// </summary>
```

```
    public partial class GeometryWindow : Window
    {
        System.Windows.Threading.DispatcherTimer dispatcherTimer;

        private GeometryModel3D geometryModel;
        private int degree = 90;

        public GeometryWindow()
        {
            InitializeComponent();

            dispatcherTimer = new System.Windows.Threading.DispatcherTimer();
            dispatcherTimer.Tick += new EventHandler(dispatcherTimer Tick);
            dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
        }

        private void rotateHead()
        {
            degree += 30;
            RotateTransform3D myRotateTransform3D = new RotateTransform3D();
            AxisAngleRotation3D myAxisAngleRotation3d = new AxisAngleRotation3D();
            myAxisAngleRotation3d.Axis = new Vector3D(0, 3, 0);
            myAxisAngleRotation3d.Angle = degree;
            myRotateTransform3D.Rotation = myAxisAngleRotation3d;
            geometryModel.Transform = myRotateTransform3D;
        }

        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            rotateHead();
        }

        public void Basic3DShapeExample()
        {
            // Declare scene objects.
            Viewport3D myViewport3D = new Viewport3D();
            Model3DGroup myModel3DGroup = new Model3DGroup();
            GeometryModel3D myGeometryModel = new GeometryModel3D();
            ModelVisual3D myModelVisual3D = new ModelVisual3D();
            // Defines the camera used to view the 3D object. In order to view the 3D object,
            // the camera must be positioned and pointed such that the object is within view
            // of the camera.
            PerspectiveCamera myPCamera = new PerspectiveCamera();

            // Specify where in the 3D scene the camera is.
            myPCamera.Position = new Point3D(0, 0, 2);

            // Specify the direction that the camera is pointing.
            myPCamera.LookDirection = new Vector3D(0, 0, -1);

            // Define camera's horizontal field of view in degrees.
            myPCamera.FieldOfView = 60;

            // Asign the camera to the viewport
            myViewport3D.Camera = myPCamera;
            // Define the lights cast in the scene. Without light, the 3D object cannot
            // be seen. Note: to illuminate an object from additional directions, create
            // additional lights.
            DirectionalLight myDirectionalLight = new DirectionalLight();
            myDirectionalLight.Color = Colors.White;
            myDirectionalLight.Direction = new Vector3D(-0.61, -0.5, -0.61);

            myModel3DGroup.Children.Add(myDirectionalLight);

            // The geometry specifes the shape of the 3D plane. In this sample, a flat sheet
            // is created.
            MeshGeometry3D myMeshGeometry3D = new MeshGeometry3D();

            // Create a collection of vertex positions for the MeshGeometry3D.
            Point3DCollection myPositionCollection = new Point3DCollection();

            MillingMachineManager myMillingMachineManager = new
MillingMachineManager(AppDomain.CurrentDomain.BaseDirectory + @"\head-vise.obj");

            foreach (Vertex v in myMillingMachineManager.GetNormalisedVertexList())
            {
                myPositionCollection.Add(new Point3D(v.x, v.y, v.z));
```

```
            }

            myMeshGeometry3D.Positions = myPositionCollection;

            // Create a collection of triangle indices for the MeshGeometry3D.
            Int32Collection myTriangleIndicesCollection = new Int32Collection();

            foreach (Face f in myMillingMachineManager.ObjectGeomertyParser.FaceList)
            {
                myTriangleIndicesCollection.Add(f.firstIndex - 1);
                myTriangleIndicesCollection.Add(f.secondIndex - 1);
                myTriangleIndicesCollection.Add(f.thirdIndex - 1);
            }

            myMeshGeometry3D.TriangleIndices = myTriangleIndicesCollection;

            // Apply the mesh to the geometry model.
            myGeometryModel.Geometry = myMeshGeometry3D;

            // The material specifies the material applied to the 3D object. In this sample a
            // linear gradient covers the surface of the 3D object.
            // Create a horizontal linear gradient with four stops.
            LinearGradientBrush myHorizontalGradient = new LinearGradientBrush();
            myHorizontalGradient.StartPoint = new Point(0, 0.5);
            myHorizontalGradient.EndPoint = new Point(1, 0.5);
            myHorizontalGradient.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
            myHorizontalGradient.GradientStops.Add(new GradientStop(Colors.Red, 0.25));
            myHorizontalGradient.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));
            myHorizontalGradient.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));

            // Define material and apply to the mesh geometries.
            DiffuseMaterial myMaterial = new DiffuseMaterial(myHorizontalGradient);
            myGeometryModel.Material = myMaterial;

            // Apply a transform to the object. In this sample, a rotation transform is
applied,
            // rendering the 3D object rotated.
            RotateTransform3D myRotateTransform3D = new RotateTransform3D();
            AxisAngleRotation3D myAxisAngleRotation3d = new AxisAngleRotation3D();
            myAxisAngleRotation3d.Axis = new Vector3D(0, 3, 0);
            myAxisAngleRotation3d.Angle = 90;
            myRotateTransform3D.Rotation = myAxisAngleRotation3d;
            myGeometryModel.Transform = myRotateTransform3D;

            // Add the geometry model to the model group.
            myModel3DGroup.Children.Add(myGeometryModel);

            geometryModel = myGeometryModel;

            // Add the group of models to the ModelVisual3d.
            myModelVisual3D.Content = myModel3DGroup;
            myViewport3D.Children.Add(myModelVisual3D);

            // Apply the viewport to the page so it will be rendered.
            this.Content = myViewport3D;

            dispatcherTimer.Start();
        }
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            Basic3DShapeExample();
        }
    }
}
```

3.4.1.3 MillingMachineCoreDLL
3.4.2. Application Interfaces

When the program is run, the following interface is shown. (*Figure 3.27*)

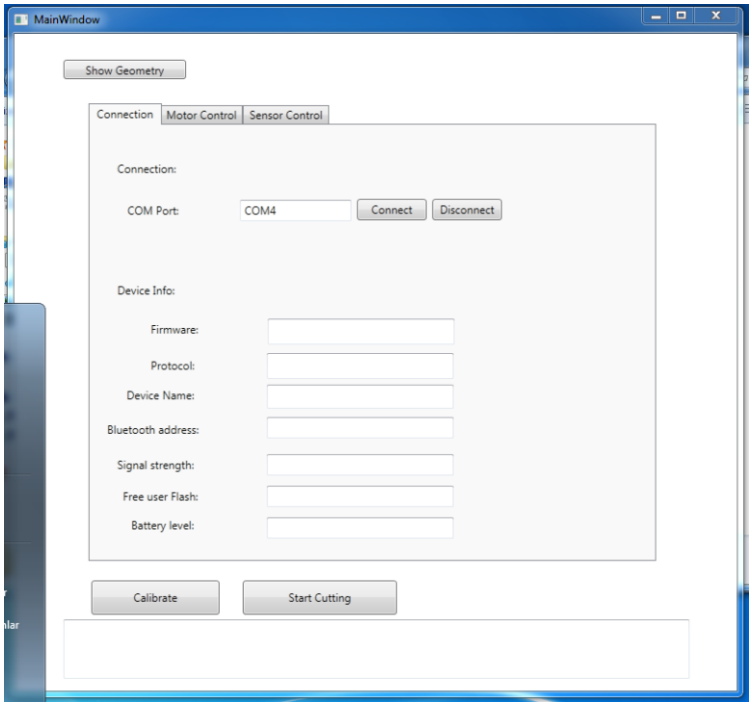


Figure 3.27- 3D milling machine interface

There are several buttons and tabs are shown in this figure. To start with **Show Geometry** button is designed to draw 3D graphic of object which was designed in Blender and ready to cut. When this button is clicked, the related object is displayed on the screen as a rotating object. (*Figure 3.28, Figure 3.29*)

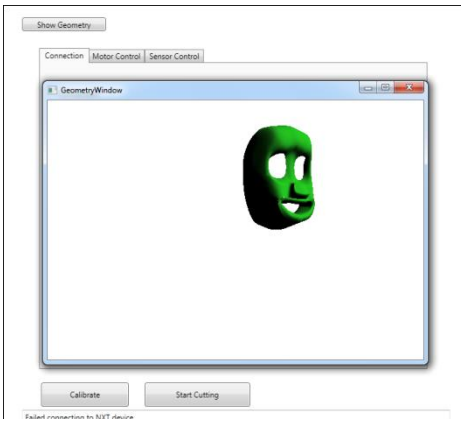


Figure 3.28 - Show Geometry_1

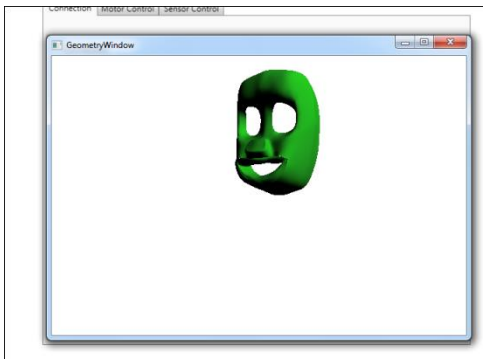


Figure 3.29 – Show Geometry_2

This button creates instance of geometry classes in "*MillingMachineGeomteryParserDll*" and parses "Wavefront .obj" file exported from Blender software as explained in previous sections. Below this button, there are three tabs available. The first one is required to make connection to the NXT brick. An example of connected state is shown on the *Figure 3.30*.

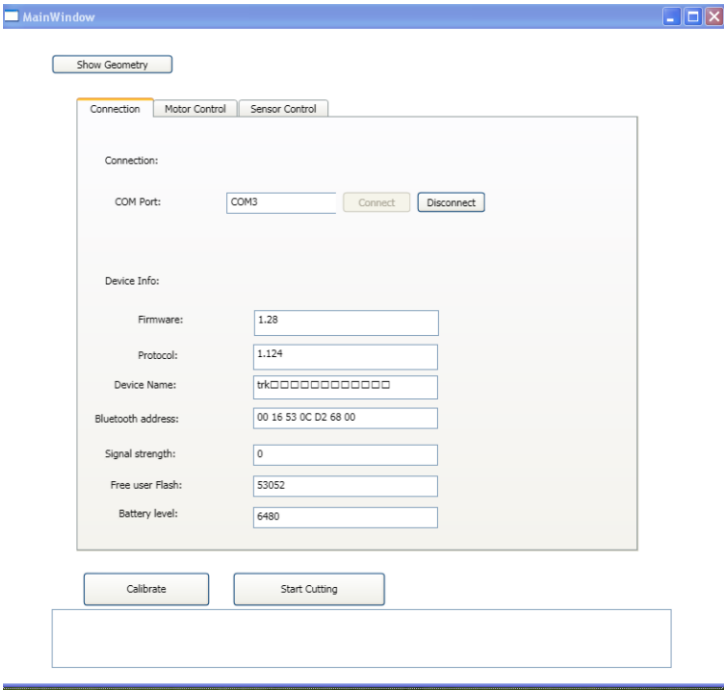


Figure 3.30 – Established connection example

The second tab; namely, **Motor Control** is responsible for controlling motors manually (*Figure 3.31*). The third tab; namely, **Sensor Control** is responsible for controlling sensors, setting sensor types and getting inputs from the sensors. (*Figure 3.32)*

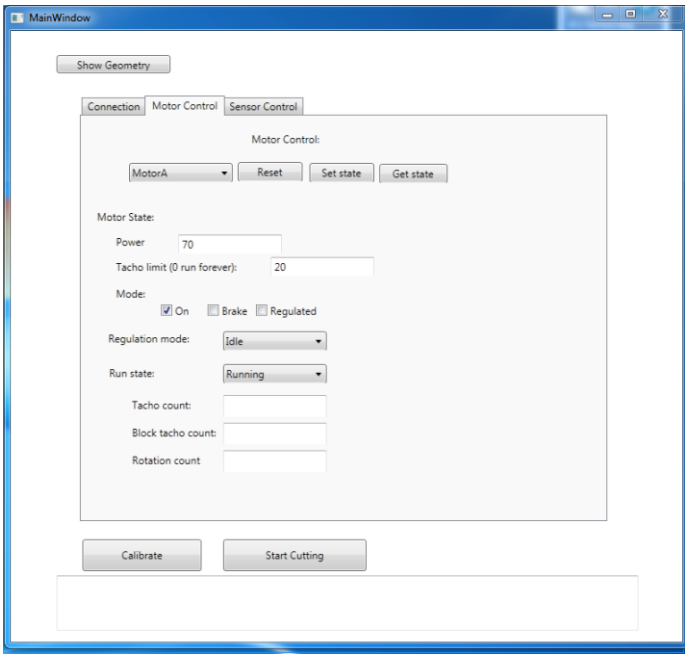


Figure 3.31 – Motor Control tab

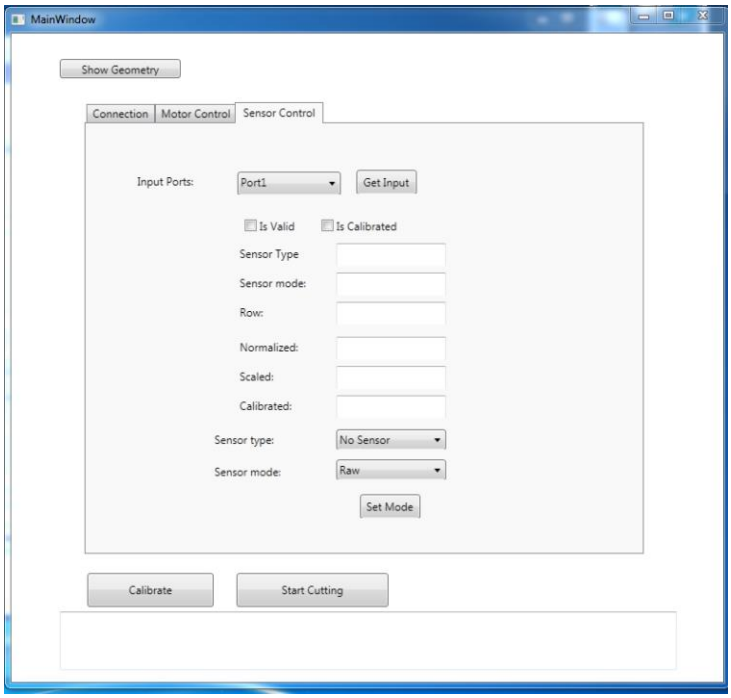


Figure 3.32 – Sensor Control tab

Under these three tabs, there are two buttons. First one; **Calibrate** button calibrates the coordinate systems of 3D objects designed in the Blender software. The second button; namely, **Start Cutting** is responsible for starting the cutting operation. This is the main button for the program and responsible for carrying out the main functionality.

3.5. Testing 3D milling machine

It has been used for testing different transformations and calculation without running the full application. Therefore, a possible damage of the newly created 3D milling machine has been avoided by limiting the machine's motor rotating values from going extreme values.

3.5.1 MillingMachineTestDLL

UnitTestMachineManager: This is the testing class to test the machine its boundary conditions. Values of "*sampleVertexList*" member variable are changed to calibrate the machine for a newly designed 3D object. The body of this class has been presented as follows:

```
[TestClass]
public class UnitTestMachineManager
{
    [TestMethod]
    public void FindMaximumAndMiniumValuesTest()
    {

        // Calibration sample
        List<Vertex> sampleVertexList = new List<Vertex>();
        sampleVertexList.Add(new Vertex { x = 3, y = 3, z = 91 });
        sampleVertexList.Add(new Vertex { x = 9, y = 4, z = 12 });
        sampleVertexList.Add(new Vertex { x = 2, y = 12, z = -3 });
        sampleVertexList.Add(new Vertex { x = -8, y = -5, z = 8 });
        sampleVertexList.Add(new Vertex { x = 4, y = -4, z = 12 });
        sampleVertexList.Add(new Vertex { x = 2, y = 0, z = 7 });
```

```
        // Act
        double maxXValue = Utility.FindMaxValue(sampleVertexList, m => m.x);
        double minXValue = Utility.FindMinValue(sampleVertexList, m => m.x);

        double maxYValue = Utility.FindMaxValue(sampleVertexList, m => m.y);
        double minYValue = Utility.FindMinValue(sampleVertexList, m => m.y);

        double maxZValue = Utility.FindMaxValue(sampleVertexList, m => m.z);
        double minZValue = Utility.FindMinValue(sampleVertexList, m => m.z);


        //Assert functions

        Assert.AreEqual(maxXValue, 9);
        Assert.AreEqual(minXValue, -8);
        Assert.AreEqual(maxYValue, 12);
        Assert.AreEqual(minYValue, -5);
        Assert.AreEqual(maxZValue, 91);
        Assert.AreEqual(minZValue, -3);
    }

}
```

4. Conclusion and future works

4.1 Reflection and Conclusion

In this project, a Lego Mindstorms® based 3D milling machine has been proposed and successfully constructed. An independent Windows Application has also been created to control this milling machine and designed objects in Blender have been cut successfully with the machine. Many Lego parts including Mindstorms® kit pieces, Lego Technic® and simple drill motor has been used in this project. All these parts have been brought together in this project to satisfy the aims defined in the first chapter.

During the execution of the projects, some obstacle has been confronted with. Initially, many prototype examples have been designed for familiarization purposes. Transformation of this familiarization knowledge to building the actual 3D machine took more time than expected. Many stability issues have to be overcome to produce a working 3D milling machine. Scalability of 3D objects which has been designed on 3D software has required very deep programming and mathematical skills. Lego Mindstorms® NXT-G software is not really for big projects and it has many crashing issues with big programming blocks. Third party communication APIs proved to be very useful during this project and extensively used to validate some initial ides, then implementing the requirements of the projects.

In this project, calibration of the machine was a really challenging task. In the Lego NXT-G, in order to calibrate the 3D milling machine, various programming blocks has been employed. However, it took a lot of time for a reasonable calibrated measurement. In C# program this action was relatively easier and more precise. Calibration method has been improved further to cut object in a certain region. In this process, Unit Test project has been used extensively to avoid damaging the machine. Because of wrong signals could rotate the motors and caused machine to go out of range. This might cause damage to the 3D milling machine.

In a general coordinate system, z axis is perpendicular to x and y axes. But as it discovered in reading coordinates from the coordinate object file (.obj), the implemented C# program takes y axes is perpendicular to x and z axis. Movement of the y axis programmed differently compare to other two axes. X and z axes are using displacement values to move motors in the right directions not full coordinates values. However, the motor controlling y axis moves the cutting part (rotating cutter) from its reference point until its full y-axis vertex value, then moves it back again up to its reference point until it hits a zone controlled by color sensor's blue color. Coordination of these coordinates systems, and their synchronizing it with blue color sensor was also a challenging task.

To sum up, a stable 3D machine has been constructed using simple Lego parts and all the aim of the project has been satisfied. Experience with Blender 3D object designing software was also a big bonus for this project. Designs are cut in milling machine according to the coordinates. If we would like to compare designs and cutting times; Face is most difficult example takes time to cut due to edge complexity. Then half sphere is not that shaped as in blender but it is quite good to cut that object in our platform. Pyramid example takes little time to cut in platform due to the shape of design comparing other designs.

The author of the project has learned a lot from working with Lego's own visual programming interface and C# program developed in Visual Studio. Both hardware and software elements in the projects made a worthwhile learning exercise

4.2. Future Works

During a lifetime of this project it has also been seen that higher level of abstraction visual programming environment provides causes some understanding problem related to robotic kits. When a relatively complex projects are implemented with the robotic kit lack of knowledge and understanding basic relationships with motor robots, sensors and Input/output characteristic causes frustrations while working with the kit. Although measuring effects of these limitations of visual programming environments is out of scope of the current thesis. This maybe studied as a future work related to visual programming environment. Surface and deep learning characteristics of students while implementing complex projects with visual and API based programming environment can also be studied.

The main future work can be improving capabilities of the 3D milling machine. Currently it can cut only small objects whose size is less than **4.8 cm * 6.8 cm** according to the design of implementation. This can be improved, and bigger objects can be cut. This project uses only wavefront(.obj) file for 3D objects. New parser can be added to the program so that the program can accept more 3D object types from different 3D design suite. Furthermore there are some platforms could be developed to cut objects 360^{o}.
Furthermore C# has been used in this project as a programming language. There are some other libraries for different language and platforms, such as Java APIs which can be employed in future works.

References

1. Akin, H. L., Meriçli, Ç, Meriçli T. and Doğrultan, E. (2009). Introduction to Autonomous Robotics with Lego Mindstorms*, WSPC - Proceedings, clawar09-Education.*
2. AForge.NET (2008). <*http://www.aforgenet.com/* >
3. Arduino (2005). <*http://arduino.cc/en/* >
4. Blender Foundation (2012). < *http://www.blender.org/* >
5. Jonathan Knudsen (2000). *Lego MindStorms: An Introduction.* O'Reilly Network*,* <*http://www.oreillynet.com/pub/a/network/2000/01/31/mindstorms/index.html*>
6. Jörgen Lindh, Thomas Holgersson (2007). Does lego training stimulate pupils ability to solve logical problems? J*ournal Computers & Education archive*, Volume 49 Issue 4, Pages 1097-1111.
7. Kim, S. H. and Jeon, J.W. (2007). Programming LEGO Mindstorms NXT with visual programming*, Control, Automation and Systems, 2007. ICCAS '07. International Conference on 17-20 Oct. 2007, Pages 2468 - 2472.*
8. LEGO MINDSTORMS:What is NXT? (2002). <*http://mindstorms.lego.com/en-us/whatisnxt/default.aspx*>
9. LEJOS, Java for Lego Mindstorms (2009). <*http://lejos.sourceforge.net/*>
10. Microsoft Robotics Developer Studio (2012). <*http://www.microsoft.com/robotics/*>
11. National Instruments' LabVIEW (2013). < *http://www.ni.com/labview/* >
12. NXC and NBC (2012).< *http://bricxcc.sourceforge.net/nbc/*>
13. Nxtprograms (2007). < *http://www.nxtprograms.com/* >
14. Perdue D., J. and Valk, L. (*2011). THE UNOFFICIAL LEGO MINDSTORMS NXT 2.0 INVENTOR'S GUIDE.* Publisher: William Pollock, No Starch Press, Inc. 38 Ringold Street, San Fransisco, CA 94103.

Appendices

Appendix A Lego Mindstorms NXT 2.0 Toolkit
The NXT Brick

Figure 1-1

The NXT **Figure 1-1** is the brain of Mindstorms robot. It's a intelligent computer-controlled lego brick that lets a Mindstorms robot come alive and perform different operations

Motor ports
The NXT has three output port attaching motors – Ports A, B and C
Sensor ports
The NXT has four input ports for attaching sensons – Ports 1, 2, 3 and 4
USB port
This port connects The NXT and computer with USB cable. Also there is wireless Blueetooth connection for uploading and downloading.
Loudspeaker
Make a program with real sounds and listen to them when program is ran.
NXT Buttons
Orange button: On/Enter/Run
Light grey arrows: Used for moving left and right in the NXT menu
Dark grey button: Clear/Go back
Technical specifications
*32-bit ARM7 microcontroller
*256 Kbytes FLASH, 64 Kbytes RAM
*8-bit AVR microcontroller
*4 Kybtes FLASH,512 Byte RAM
*Bluetooth wireless communication (Bluetooth Class II V2.0 compliant)
*USB full speed port (12 Mbit/s)
*4 input ports, 6-wire cable digital platform (One port includes a IEC 61158 Type 4/EN 50 170 compliant expansion port for future use)

Color Sensor

Figure 1-2

The Color (see figure 1-2) sensor is one of the sensors that gives robot vision (the Ultrasonic Sensor is the other)

The Color Sensor actually has three different functions in one. The Color Sensor enables robot to distinguish between colours and light and dark. It can detect 6 different colors, read the light intensity in a room and measure the light intensity of colored surfaces. The Color Sensor can also be used as a Color Lamp.

Touch Sensor

Figure 1-3

The Touch (see Figure 1-3) Sensor gives robot a sense of touch. The Touch Sensor detects when it is being pressed by something and when it is released again.

Ultrasonic Sensor

Figure 1-4

The Ultrasonic Sensor (see figure 1-4) is one of the two sensors that give robot vision [The Light Sensor is the other]. The Ultrasonic Sensor enables robot to see and detect objects. It is also can be used to make robot avoid obstacles, sense and measure distance, detect movement.

The Ultrasonic Sensor measures distance in centimeters and in inches. It is able to measure distances from 0 to 255 centimeters with a precision of +/- 3 cm.

The Ultrasonic Sensor uses the same scientific principle as bat: it measures distance by calculating the time it takes for a sound wave to hit an object and return – just like an echo.

Large sized objects with hard surfaces return the best readings. Objects made of soft fabric or those are curved [like a ball] or are very thin or small can be difficult for sensor to detect.

Serve Motors

Figure 1-5

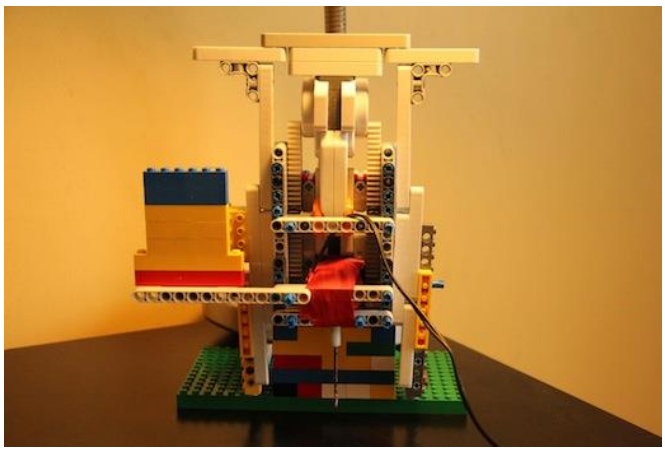
The three Servo Motors (see figure 1-5) give robot the ability to move. If you use the Move block in the Lego Mindstorms NXT software to program motors, the two motors will automatically synchronize, so that robot will move in a straight line.

Built-in Rotation Sensor

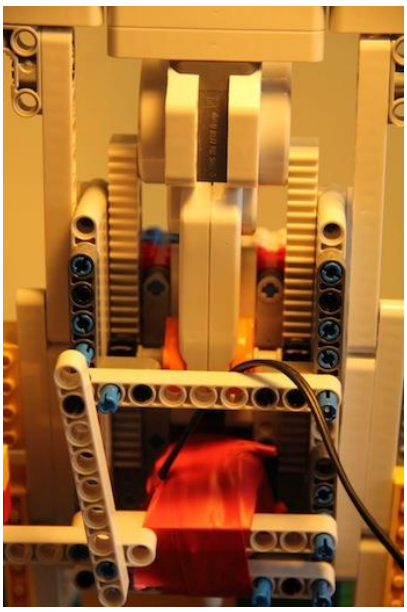
Each motor has a built-in Rotation Sensor. This lets you control robot's movements precisely. The Rotation Sensor measures motor rotations in degrees or full rotations [accuracy of +/- one degree]. One rotation is equal to 360 degrees, so if it is set a motor to turn 180 degrees, its output shaft will make half a turn.

The built-in Rotation Sensor in each motor also has different speeds for motors [by setting different power parameters in the software].

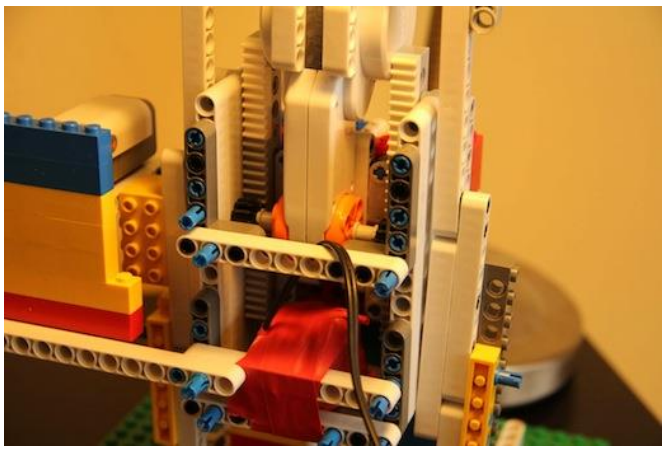
Appendix B Detailed pictures from platform
Part-1:Y-axis



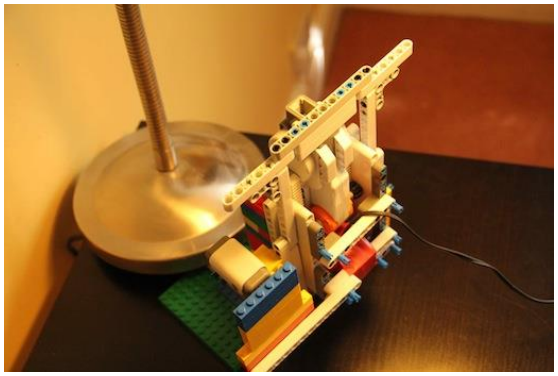
Picture B.1- Front view



Picture B.2- Front view



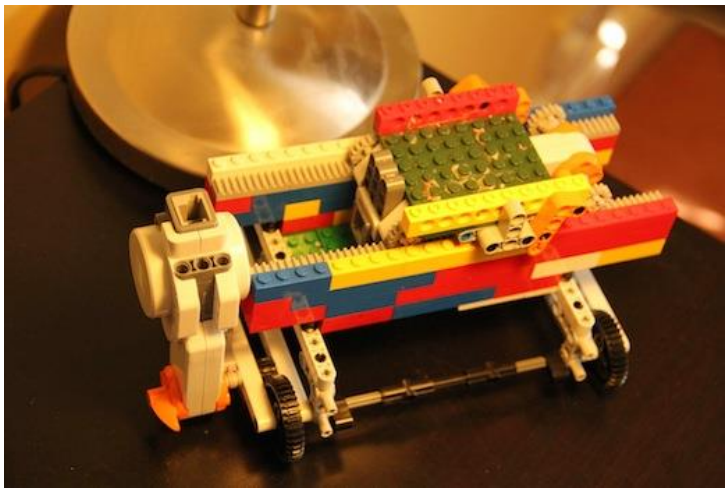
Picture B.3- Front view



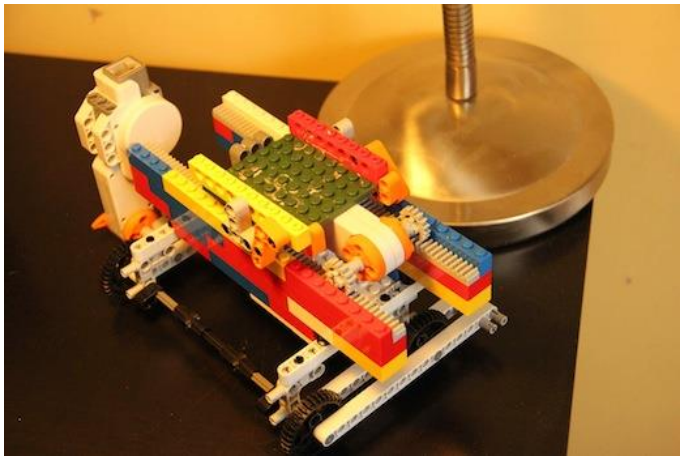
Picture B.4- Front-Up view

Picture B.5- Back view

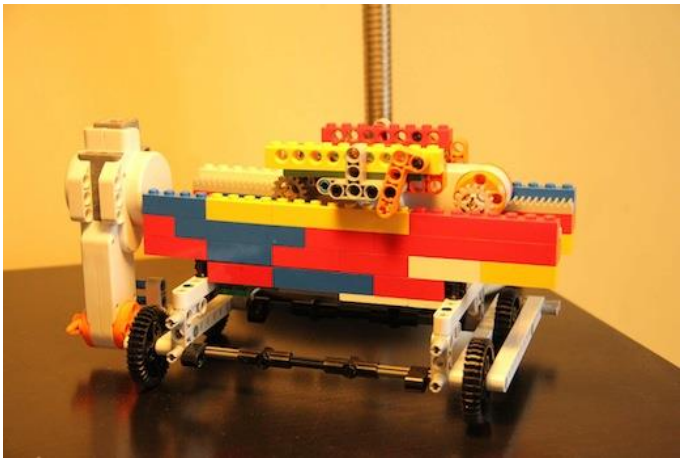
Part-2: X-axis and Z-axis together



Picture B.6-Left Side view

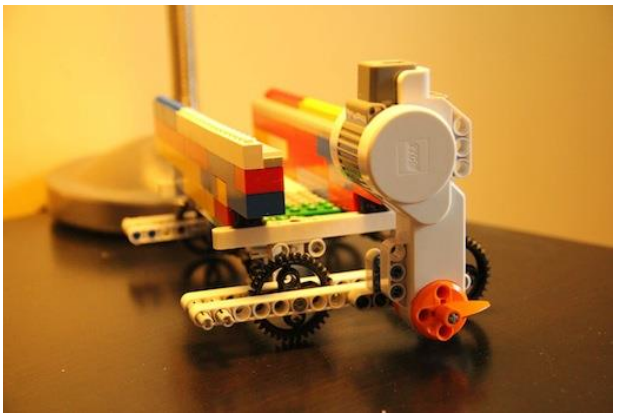


Picture B.7-Right Side view

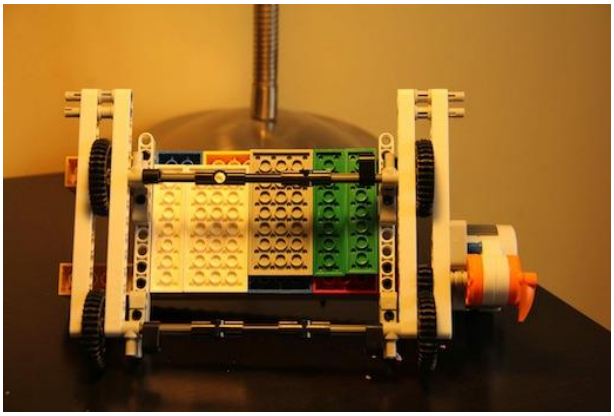


Picture B.8-Front view

Part-3: Z-axis

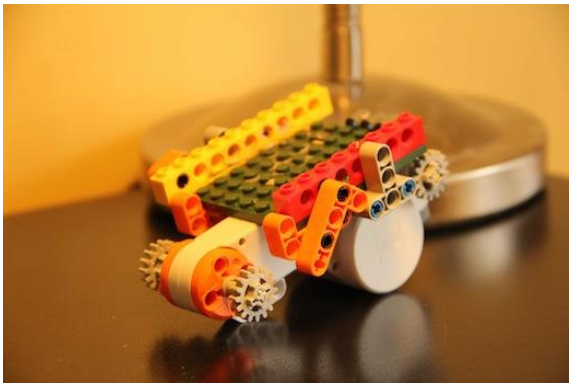


Picture B.9-Back view

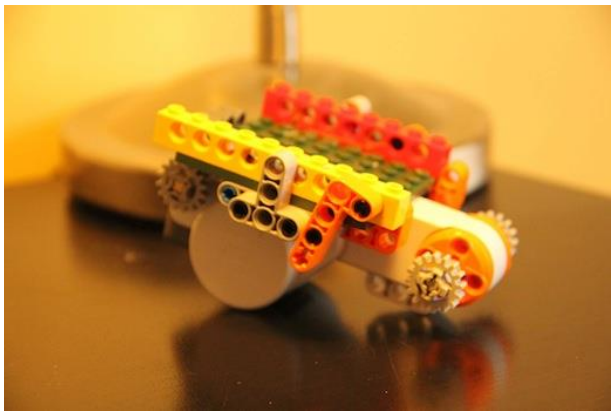


Picture B.10-Bottom view

Part-4: X-axis

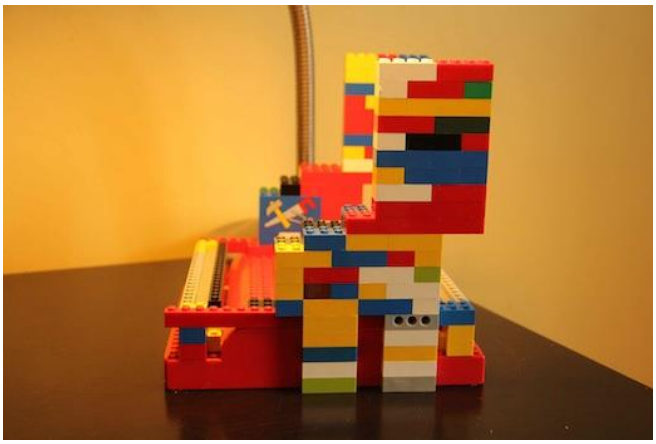


Picture B.11-Right Side view



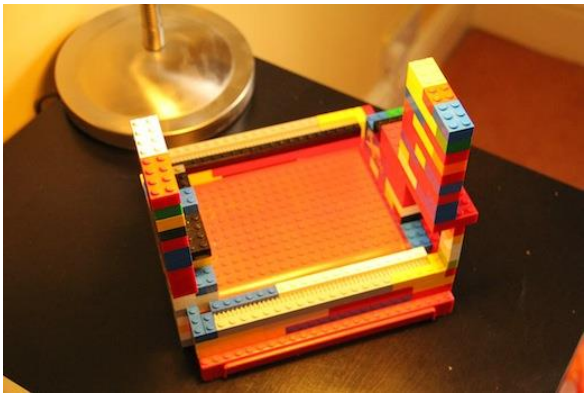
Picture B.12-Left Side view

Part-5: Platform



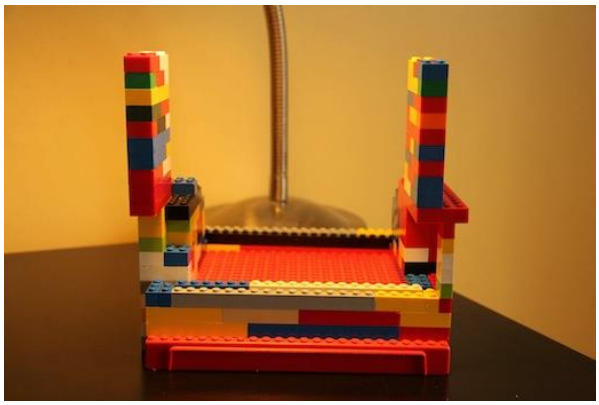
Picture B.13-Left view

Picture B.14-Right view



Picture B.15-Up view

Picture B.16-Up view



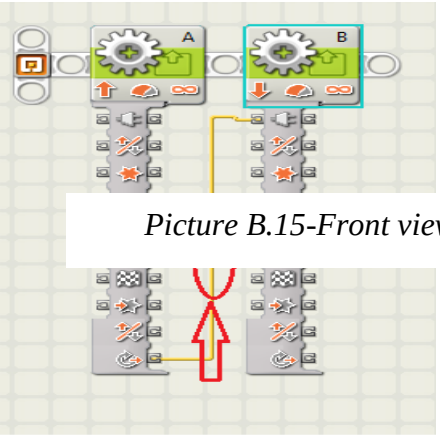
Picture B.17-Up view

Appendix C NXT-G programming

NXT 2.0 set is becoming proficient in programming your creations to do what you want them to do. Basic programming skills would be enough to programming NXT-G to perform basic tasks. But more complex tasks require more advanced level of programming skills.

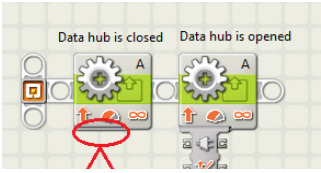
Data wires

You can use *data wires* to transmit information between programming blocks. With the exception of the Wait block, all programming blocks can use data wires to one degree or another and a number of blocks require the use of *data wires.* (Figure 1-1)



Picture B.15-Front view

Figure 1-1: Data wires transmit information between programming blocks



Picture B.16-Back view

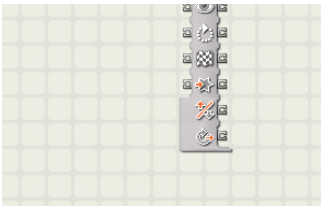


Figure 1-2: A block's data hub can be accessed by clicking the small tab in the lower-left corner of the block

First and foremost, the foundation of data wires is the *data hub* (Figure 1-2)
Along the length of each data hub is one or more data plugs, which come in two main types: *input plugs* and output *plugs* (Figure 1-3)

Data plug characteristics

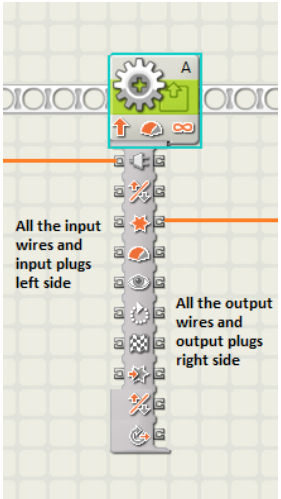


Figure 1-3: Input data wires and plugs and Output data wires and plugs.

First, a data plug relates to particular aspects of its block, symbolized by an icon on the block's data hub.
Second, a data plug uses a specific data type: *number, logic, text.*
Third, a data plug often accepts only a certain range of *values*.

Note !: The Wait, Loop and Switch blocks don't have data hubs, but the Loop block and Switch block do have data plugs that appear for certain configurations.

The wire path

When a piece of data initially leaves an output plug and travels over a data wire to one or more blocks, it's following a *wire path*.

Two important guidelines of create functional wire paths:

First, every wire path must connect to at least one output plug and one input plug.
Second, you can extend a wire path to reach more than one block using corresponding pairs of input and output plugs. (Figure 1-4)

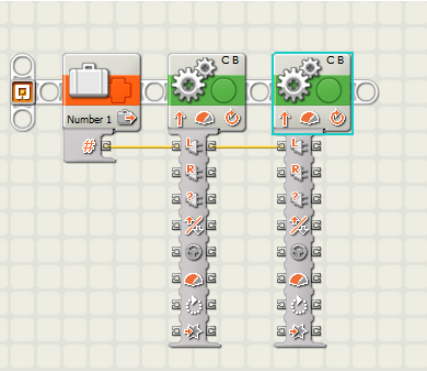


Figure 1-4: Sample wire path

Note: You cannot use an output plug if it has a corresponding input plug without a data wire connected to it; the output plug serves only to transmit input data to other blocks.

Transmitting the data types
A data wire must connect plugs of the same data type. The data wires are color-coded. Data wires that carry logic data are green, data wires that carry text data are orange, and "broken" data wires are gray.

Broken data wires
A data wire "breaks" when you've made an invalid connection between two data plugs, and you cannot download a program to your NXT if that program contains broken wires. Three type errors cause breaks:
Data type mismatch, missing input(Figure 1-5), or too many inputs.

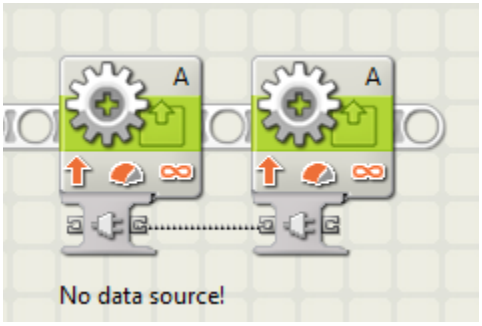


Figure 1-5: This broken data wire is missing an input

Note: The NXT-G documentation on the software discusses a fourth type of error, a *cycle*, which occurs when a wire path "visits the same block twice." However, the software prevents you from creating cycles by not allowing certain data wire connections.

The Complete Palette

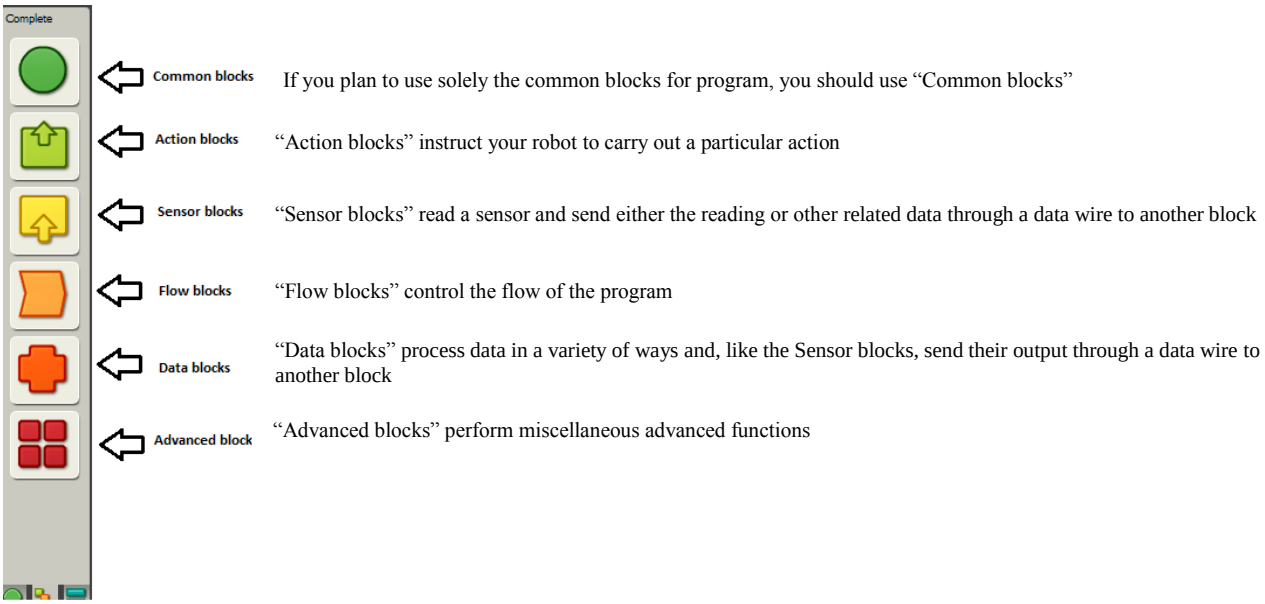


Figure 1-6: The Complete palette includes six categories of programming blocks

The Common blocks

Figure 1-7: The Common block's sub-palette includes seven common blocks, the Move block, the Record/Play block, the Sound block, the Display block, the Wait block, the Loop block and, the Switch block

The Action blocks

Figure 1-8: The Action block's sub-palette includes the Motor block, the Sound block, the display block, the Send Message block, and the Color Lamp Block

The Sensor blocks

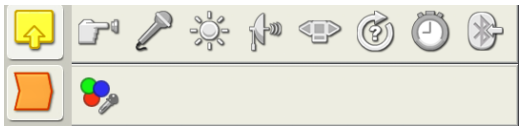


Figure 1-9 The Sensor block's sub-palette includes nine sensor blocks, the Touch Sensor block, Sound Sensor block, Light Sensor block, Ultrasonic Sensor block, NXT Buttons Sensor block, Rotation Sensor block, Timer Block, Receive Message block and Color Sensor block

Note: The LEGO MINDSTORMS NXT 2.0 set (#8547) does not include sound or light sensor, you have to purchase these sensors if you want to use these blocks.

The Flow blocks

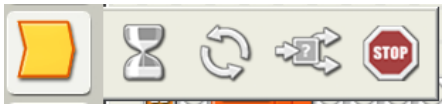


Figure 1-10: The Flow block's sub-palette includes the Wait block, Loop block. Switch block, and Stop block

The Data blocks

Figure 1-11: The Data block's sub-palette includes seven data blocks, the Logic block, Math block, Compare block, Range block, Random block, Variable block, and Constant block

The Advanced blocks

Figure 1-12: The Advanced block's sub-palette includes seven advanced blocks, the Number to Text block, Text block, Keep Alive block, File Access block, Calibration block, Reset Motor block, and Bluetooth Connection block

The basis of My Blocks is the standard programming blocks.(Represents one or more standard blocks and execute those blocks when it's executed)

A Web Block is simply a My Block that someone else has created and that you've presumably downloaded from the internet