State of the art in continuous voice recognition and command processing

Lucian Tuca, Faculty of Computer Science Iasi

# Abstract

Recent years have witnessed significant progress in voice recognition and voice commands. Given an example vocal input the voice recognition algorithm transforms it to readable string. In this state-of-the-art report, we aim to achieve these goals: (1) briefly describe the method of voice recognition used within the application, (2) describe the mechanism used for processing and executing the commands (3) provide a hawk eye view over the entire project.

# Introduction

Voice Geometry Painter is an application that allows people (especially Mathematics teachers and pupils, but not only) to draw geometric shapes by speaking certain commands.

The purpose of the application is to allow people that have some motion disabilities to draw and manipulate geometric shapes, helping them to study geometry. We think this could be a great help for mathematics teachers that have a motion disability and cannot draw/write at the whiteboard.

The interaction with the application is simple, based on short voice commands like *draw <shape name> <simple parameters>* for drawing, and even simpler commands for other operations, like *clear board*.

# Voice Recognition

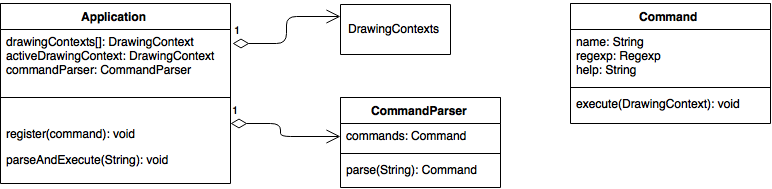
The speech recognition component uses an instance of WebkitSpeechRecognition engine, subscribing a set of callbacks for recognition events like *OnResult* or *OnEnd*.

In addition, having a set of commands as input, the component creates a grammar, which is later used as a state machine in the recognition step. This functionality is implemented in CommandParserStream.js.

The state machine helps the application to get a bigger recognition rate by allowing “noise” between words that are actually forming the command. After a certain part of the command string was recognized the state machine ignores words that are not part of the command, eliminates them and only keeps what is needed to form a possible correct command.

# Input parsing and command execution

Given the fact that now we have the voice recognized string, we can start analyzing it in order to see if it’s a valid command to be executed. The UML diagram below (Figure 1) shows the classes used to achieve this and their structure.



Figure

The main entity of the project is the Application class. It contains two other major classes: the CommandParser and the ActiveDrawingContext. For a good and safe implementation the command design pattern was used.

The CommandParser class is used to both register new commands and to execute them. To execute a command its registration it’s first needed. This is achieved by commandParser.register(CleanBoardCommand).

The next step is to execute a command. When the user says “Clear board”, the Voice Recognition module provides the recognized string to the app.

Inside the parseAndExecute(String) method within the main application class, the command parser is called with the recognized string as a parameter to check if what the user spoke matches any of the registered commands. If the vocal input is recognized a CleanBoardCommand object is returned.

To see if the command string matches any of the registered commands, the CommandParser tries to match it to every command that it has registered. If a command matches it returns an instance of it.

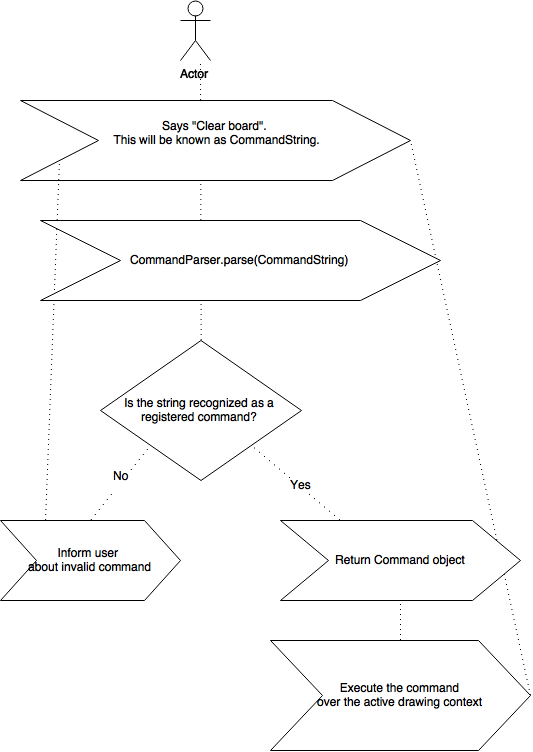
Now that we know we have recognized a vocal command it’s time to execute it. This is achieved by calling the execute method of the returned object with the activeDrawingContext as parameter.

The fact that all the command logic is implemented inside its execute method which uses the drawingContext received as parameter provides the app an incredible adaptability and extensibility.

*Adaptability*: we refer to the fact that any drawing context can be used. For example, providing we have an interface, that a certain drawing context implements the commands we’ll work out of the box, as they are just calling methods that the interface forces the classes to implement.

*Extensibility*: given the fact that every command has its own class and implementation allows the developers to add or remove any commands by any criteria that is needed. For example this criteria can be the student/teacher roles.

The steps detailed above can be better understood in the flow diagram below. (Figure 2)



Figure

# Drawing

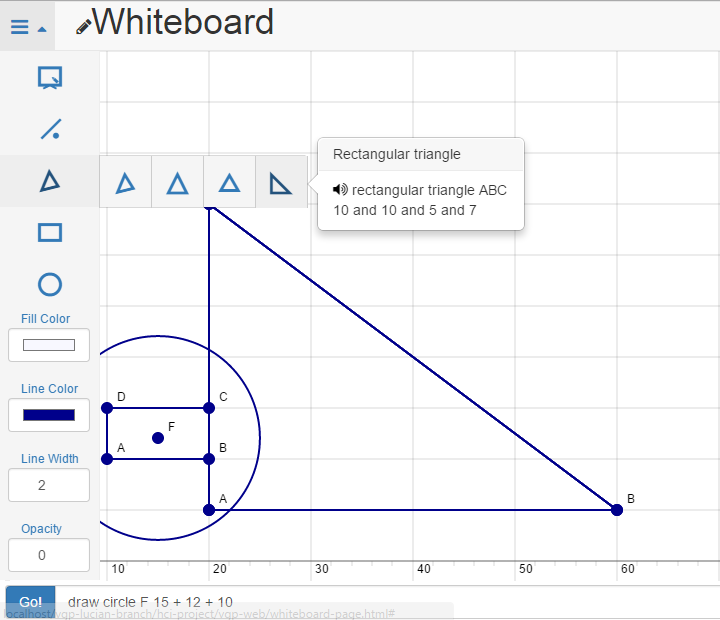
The drawing functionality is implemented in DrawingContext.js file and uses jxgraph library.

Inside the DrawingContext class there are specific drawing methods for each of the figures that are supported by the application.

The application is able to create any new figure and build a very complex set o shapes. Unfortunately at the moment the application can’t interpret any 3 random points on the board as a valid triangle. For example we have a rectangle ABCD with AC as diagonal. The application cannot provide the ABC triangle details even if they form a valid one. The triangle must be re-drawn in order for the application to memorize it.

To draw a certain figure the user must provide a complete set of parameters (including point notations). This is an area that can be vastly improved as at the moment the user can draw multiple ‘A’ points.

The application is made in such way that a person cannot draw an invalid figure. For example the user cannot draw an invalid equilateral triangle as the parameters that are accepted by the drawing method (and command) are the location of the A point, and the side length. (the direction of drawing is set by default to anti-clockwise).



Figure

# Future Work

As said above, given the great flexibility and extensibility of the modules, the app can be developed to a much more larger scale. The main area of improvement is the drawing and the board on which figures are drawn.

By this we refer to the fact that the application should be able to link several, already drawn points, to a known figure, and be able to provide details about it.

Also, a very nice to have feature, would be the capability of drawing certain specific lines for a given figure. For example, the app should be able to automatically draw for a triangle its bisectors, medians and heights.

Another great improvement to the app would be the addition of 3D figures. This feature should only be added after the 2D is complete and provides a good quality.

# Conclusion

The app is a great proof-of-concept that reveals the incredible power of voice interaction with a device. Thanks to the Google free voice recognition API we managed to implement an application that has all the chances and possibilities to grow and extend. The modules presented in the above chapters are providing a good base for anyone looking to extend the application given its simple but powerful architecture.