Bilkent University

GE401

Progress Report for CS Role Version V

Assistant Eye

Team 9

Behiç Buğra Bacanlı

Mevlüt Türker Garip

Burak Işık

Yunus Burak Suçsuz

Melik Koray Üster

Çağlar Varan

Date

13.04.2012

Table of Contents

Introduction

This document aims to provide a detailed description of the progress on structure of the overall system and its components. The eye-detection system is designed for the wheelchair used by completely paralyzed people. The name of the product produced by our company EyeCue is assistant eye. The content of this document will partly be constructed on the previous software module and task specification document. In this document, the software description will start with a system overview supported by software architecture diagram which depicts the input output interaction between subcomponents of the system. Afterwards, the software modules in these components will be explained with their characteristics and constraints. Finally, the low-level software design will be introduced, supported by algorithm flowcharts, pseudocodes and complexity analysis of the software. To create a better sense of the software, use case scenarios and their test results will be included at the end.

Proposed System Overview

The software continuously captures images of the eye one after another using a camera. After applying several transformations, which will be explained in detail in the next sections, motion detector will be activated. The motion will be detected by comparing two consequent images in pixel levels. Since the pixel-level motion detection is highly sensitive to even a little movements on the face, the prior transformations will decrease the sensitivity by manipulating the image properties such as its contrast and sharpness of the moving edges.

The camera will be placed near to the eye to detect only the movements of the eye. The calibration of the software will play a very important role for the accuracy of the software. The calibration values includes the threshold values of the distance of the eye from the center and the threshold value of the number of pixels required to be detected in order to trigger the motion detection. In other words, these threshold values will be changed according to the placement of the camera. If it is placed very near to eye, the number of pixels threshold and distance of the eye threshold will be set to a high value since lots of pixels will be detected and the distance, which the eye will move, will be significantly large.

After the eye motion is detected, the software will transmit the corresponding signals to the engine driver circuit through serial port. The software uses a highly developed open source image processing library called OpenCV. With the help of this library, the system managed to response to the user interaction in real-time speed.

High Level Software Design and Underlying Decisions

The software is implemented based on the well-known image processing library called OpenCV. The OpenCV is a open source library that is written in C and can be integrated into the Java applications by using another library called JavaCV. Since the functions of the OpenCV is installed in the dll system files, its functions can only be invoked by using Java native interface. JavaCV is the library which is written especially for this purpose. It offers the exact functions of the OpenCV and delegates each function call in the Java application to the corresponding function in the native binaries of the OpenCV dll system files.

The software first applies Gaussian transform to the captured image, which basically blurs it with all the edges on the face inside. With this transformation, the high sensitivity of the motion detector will be decreased to the reasonable level for the program accuracy. In other words, all the unnecessary small movements in pixel level will be tolerated.

After this procedure, all the captured images will be converted from RGB to the grayscale in order to ease the job of the motion detector. In this way, the detector will not have to deal with the complexity of a colored scale. After these transformations, the motion detector takes the absolute difference of two consequent images. That gives the pixel locations of the parts which moved; those pixels will be present in the image array and other pixels will be zeroed by the function diff. After that, the software will apply another transformation that converts this array to a binary array where existing pixels will be set to 255 and others to 0.

Last procedure will iterate through this array and find the mass center of the existing pixels. If the center is located left more than distance threshold, that means the eye moved to the right and the same logic exists with the other eye movement directions.

Task Decomposition and Software Components

There are three main tasks of the software for a successful detection:

Appropriate Transformations

For the software to detect the motion successfully, some image transformations are required. As also described in the previous sections, first transformation for this purpose is the Gaussian transformation, which blurs the image, in order to calibrate the sensitivity of the motion detector to a reasonable level. The other transformation is the RGB to Grayscale conversion to reduce the complexity of a colored scale. This task is performed successfully by the OpenCV component of the software and provides a significant performance improvement.

Motion Detection

After the transformations, motion detector component takes the absolute difference of the two consequent images and keeps the pixels moved in the image array. After the absolute difference, detector applies the final transformation that will set the values of the existent pixels to 255 and others to zero. This result array is the most important output of the software because the next task, which is the detection of the eye movement, is dependent on this output array.

Eye Movement Detection

This task uses the output array of the Motion Detection task. It iterates through this array and finds the mass center of the pixels whose values are 255. Since the camera is placed very near to the eye, the mass center of those pixels gives the exact location of the eye. Finally, software identifies the eye movement direction by comparing this mass center of the eye with the center coordinates of the image and sends the movement signal to the engine driver circuit through serial communication.

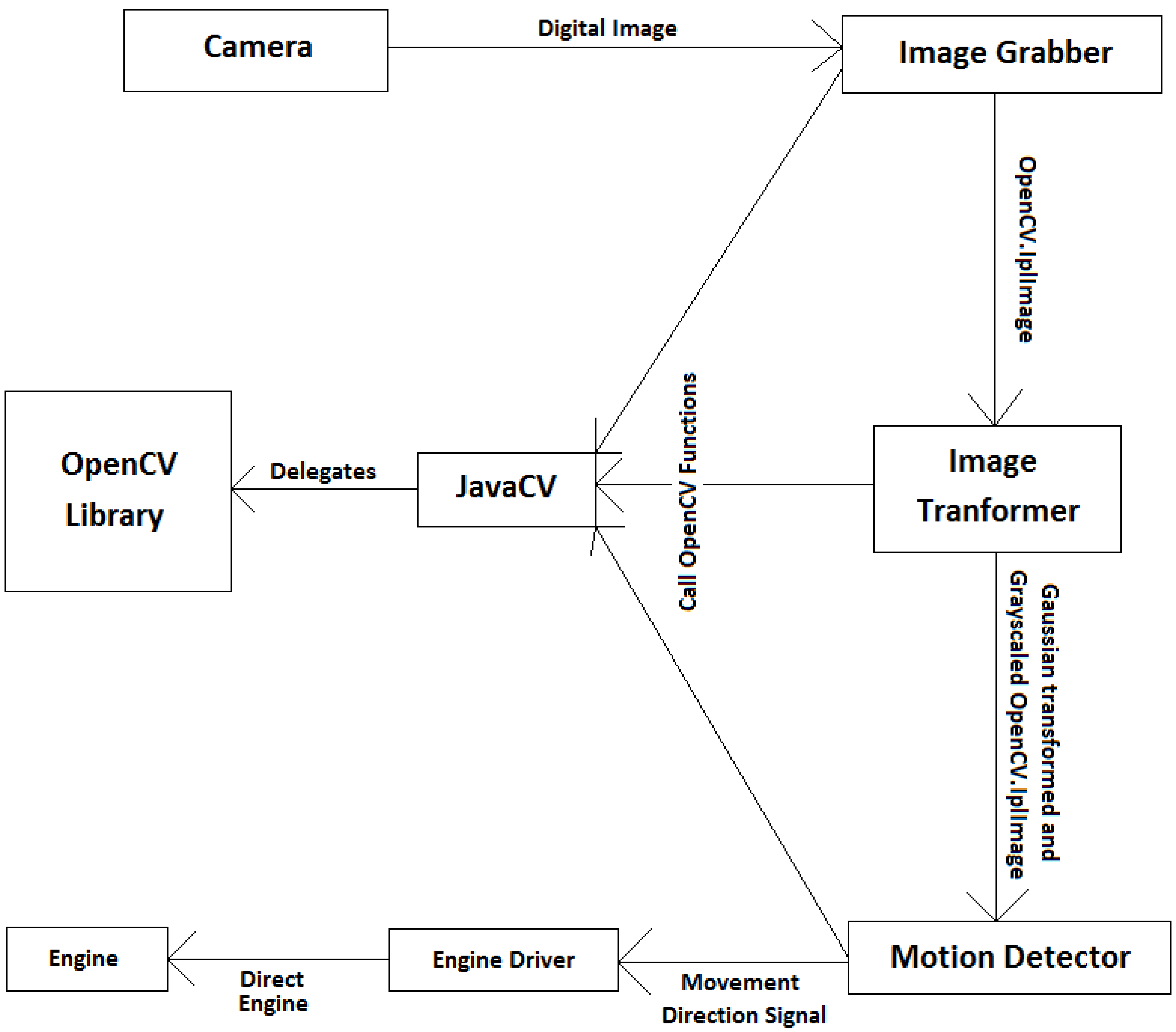
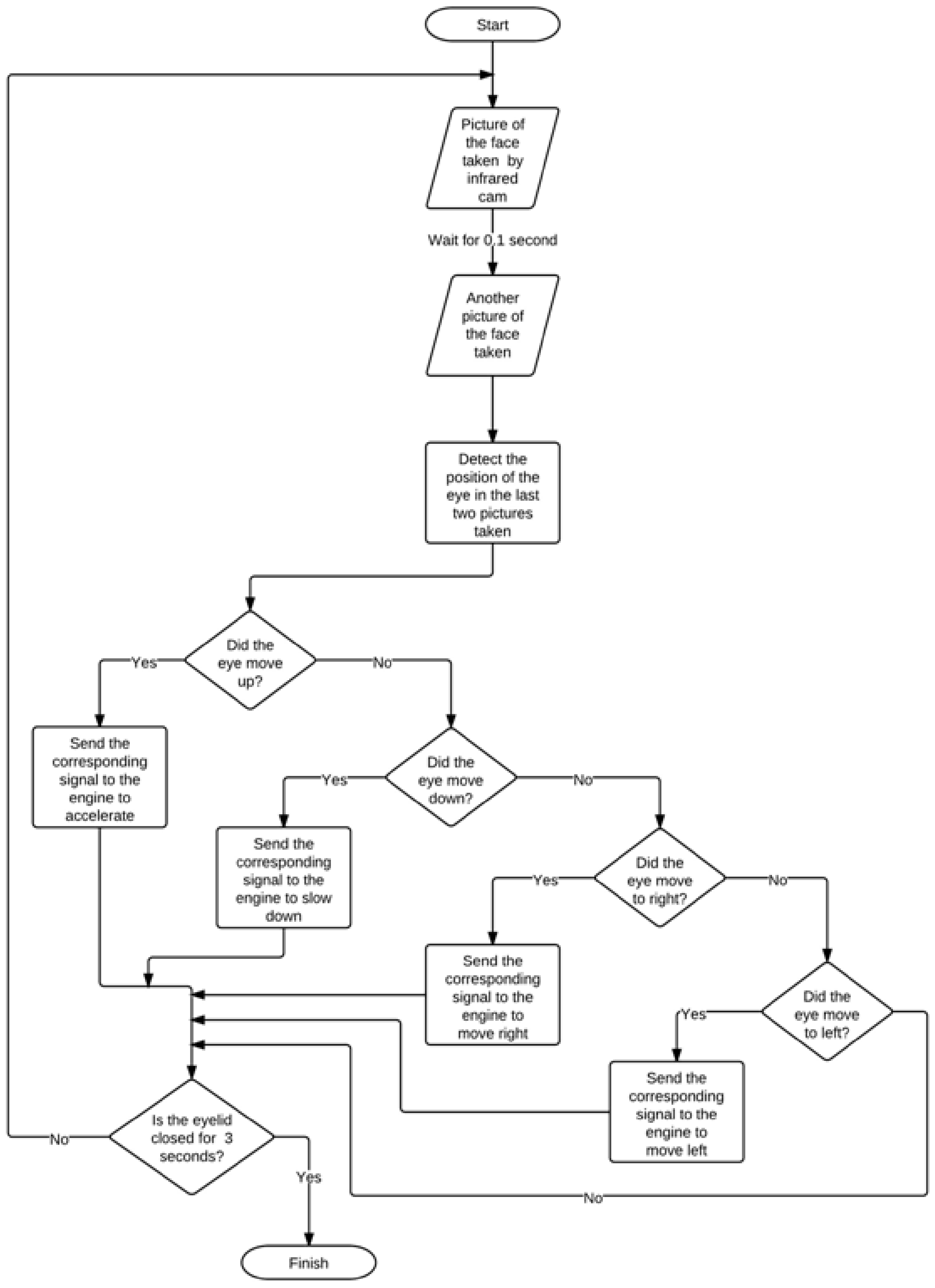


Figure 1 - Software Components with Their Interactions

Low Level Software Design



Start
Picture of the face taken by infrared cam
Wait for 0,1 second
Another picture of the face taken
Detect the position of the eye in the last two pictures taken
Did the eye move up?
Yes
No
Send the corresponding signal to the engine to accelerate
Did the eye move down?
Yes
No
Send the corresponding signal to the engine to slow down
Did the eye move to right?
Yes
No
Send the corresponding signal to the engine to move right
Did the eye move to left?
Yes
No
Send the corresponding signal to the engine to move left
Is the eyelid closed for 3 seconds?
No
Yes
Finish

Figure 2 - Flowchart of the Software

Algorithm Pseudocodes

```
void main(...){
    OpenCVFrameGrabber grabber = new OpenCVFrameGrabber(1);
    grabber.start();
    IplImage frame = grabber.grab();
    int[] image array = new int[480X640];
    String eyePosition="ORTA";
    while(frame is visible && (frame = grabber.grab()) is not null) {
        gaussianTransform(frame);
        RGBtoGray(frame);
        if(previousFrame is not null) {
            absoluteDifference(frame, previousFrame, diff);
            thresholdToBinary(diff);
            showImage(diff);
            diff.getBufferedImage().getRaster().getPixels(..., array);
            String result = detectEyeMotion(array);
            if(result is not null)
                eyePosition = result;
            System.out.println(eyePosition);
        }
    }
    grabber.stop();
}
```

```
String detectEyeMotion(int[] array) {
        int column = 0;
        int row = 0;
        int count = 0;
        for(iterate array.length times){
                if(array[i]==255){
                        column += (i % 640);
                        row += (i / 480);
                        count++;
                }
        }
        if(count < 5000)                                // Apply number of pixels threshold
                return null;
        else{
                row /= count;                           // Find the center mass of the pixels
                column /= count;
                if(column > 370)
                        return "SOL";
                else if(column < 270)                   // Detect the eye direction according to the
                        return "SAG";                   // center mass using distance thresholds
                else if(row>350)
                        return "ASAGI";
                else if(row<180)
                        return "YUKARI";
                else
                        return "ORTA";
      }
}
```

Complexity Analysis

Using the OpenCV library has increased performance of the software significantly since it uses optimized image analysis techniques for the common functions. With this way, software manages to work in linear time which is O(n). I did not consider the main while loop where the camera captures the images because it does not affect the response time and continues infinitely as long as the program is running. OpenCV analyzes the images in linear time and my eye motion detector iterates through the image array with a single for loop; therefore, also works in linear time.

Potential Limitations and Problems

The most significant problem is the calibration. The success of the software highly depends on the success in setting the correct threshold values. The correct calibration can only be done when the camera is placed and held still from the head.

Another problem is the eye blinking. Since the software detects all the motions, eye blinking often dominates the detection of the eye. For example, if the user look right and blinks, the mass center of the moving part will probably be at the center but user actually looked right.

These problems will be handled successfully in the integration and testing; because, only in that time, we will be able to integrate the product, place the camera on the head and make it hold still. During that progress, the threshold values can be set accordingly and the problem can be eliminated. That is why calibration is the most important task before the product is completed.

Use-case Scenarios

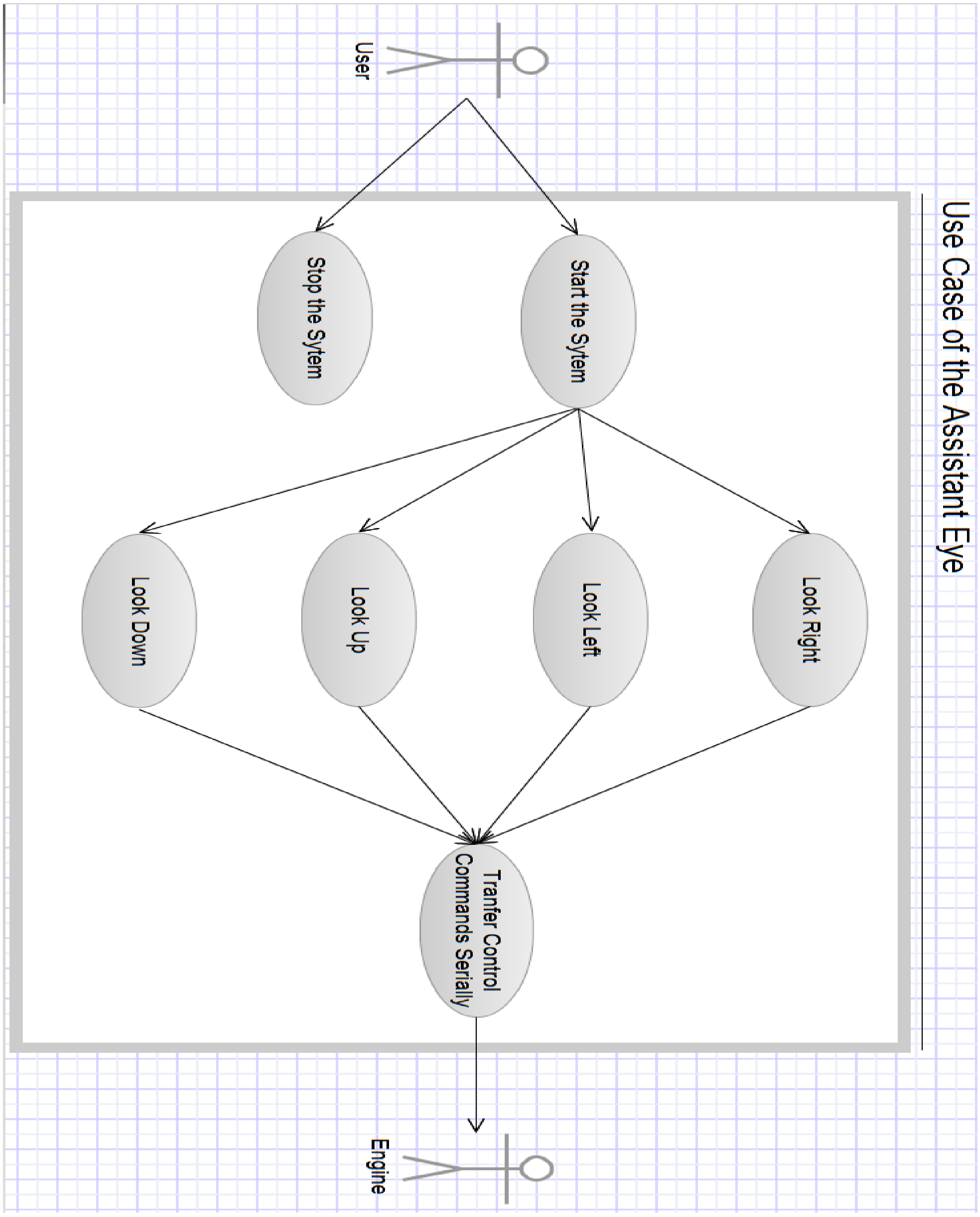
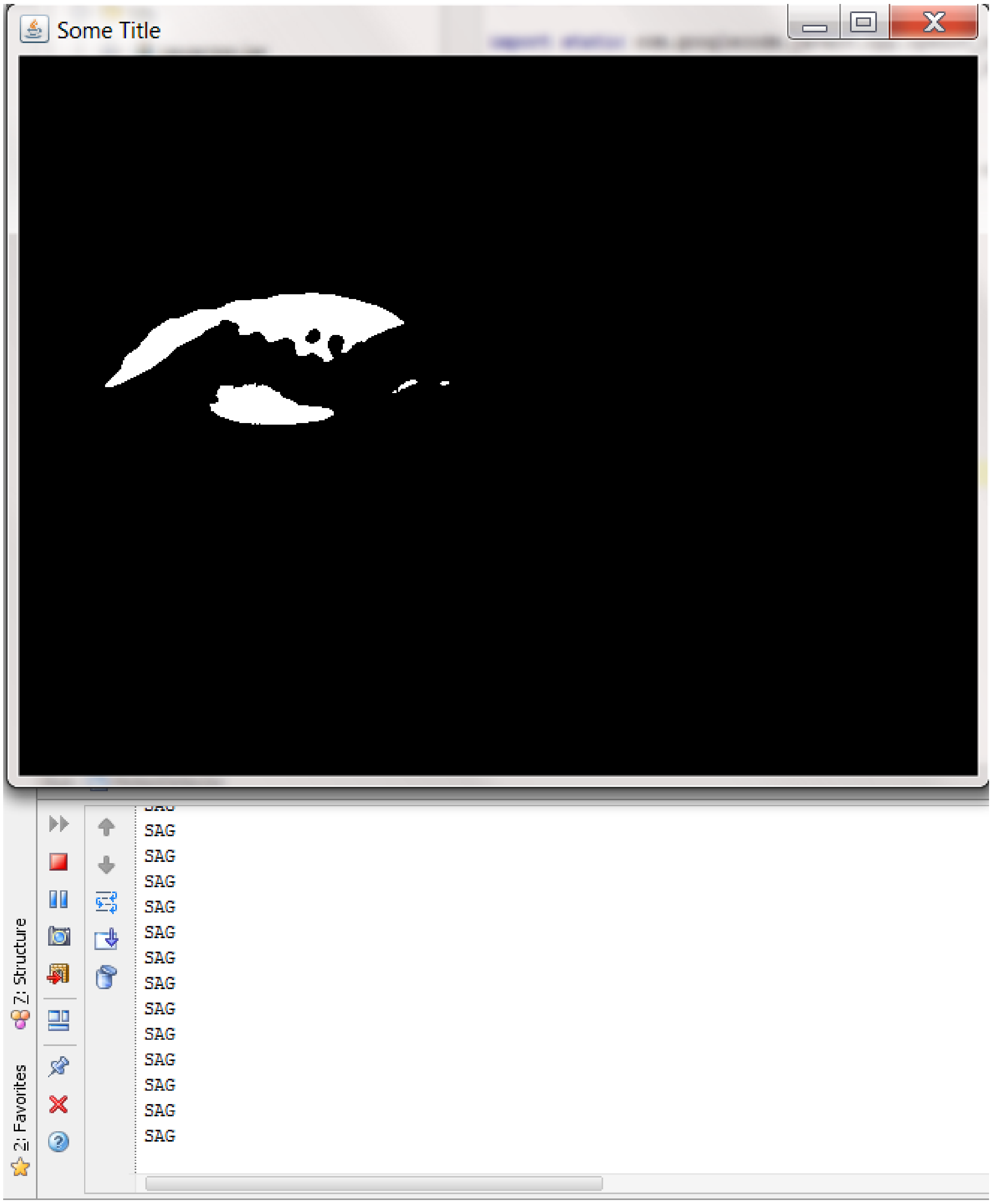


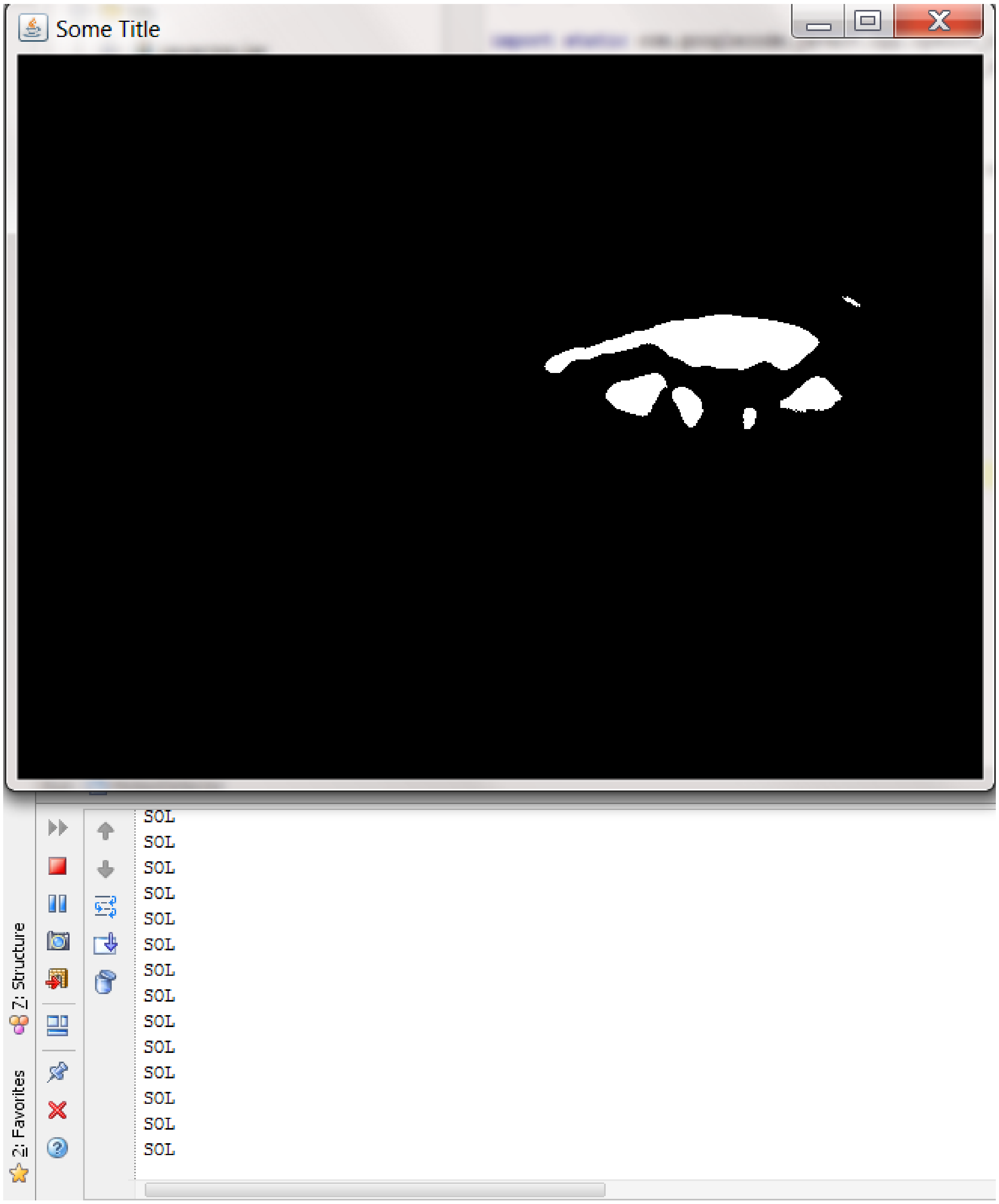
Figure 3 - Use Case Diagram of the Software

Software Test Results with the Use Cases



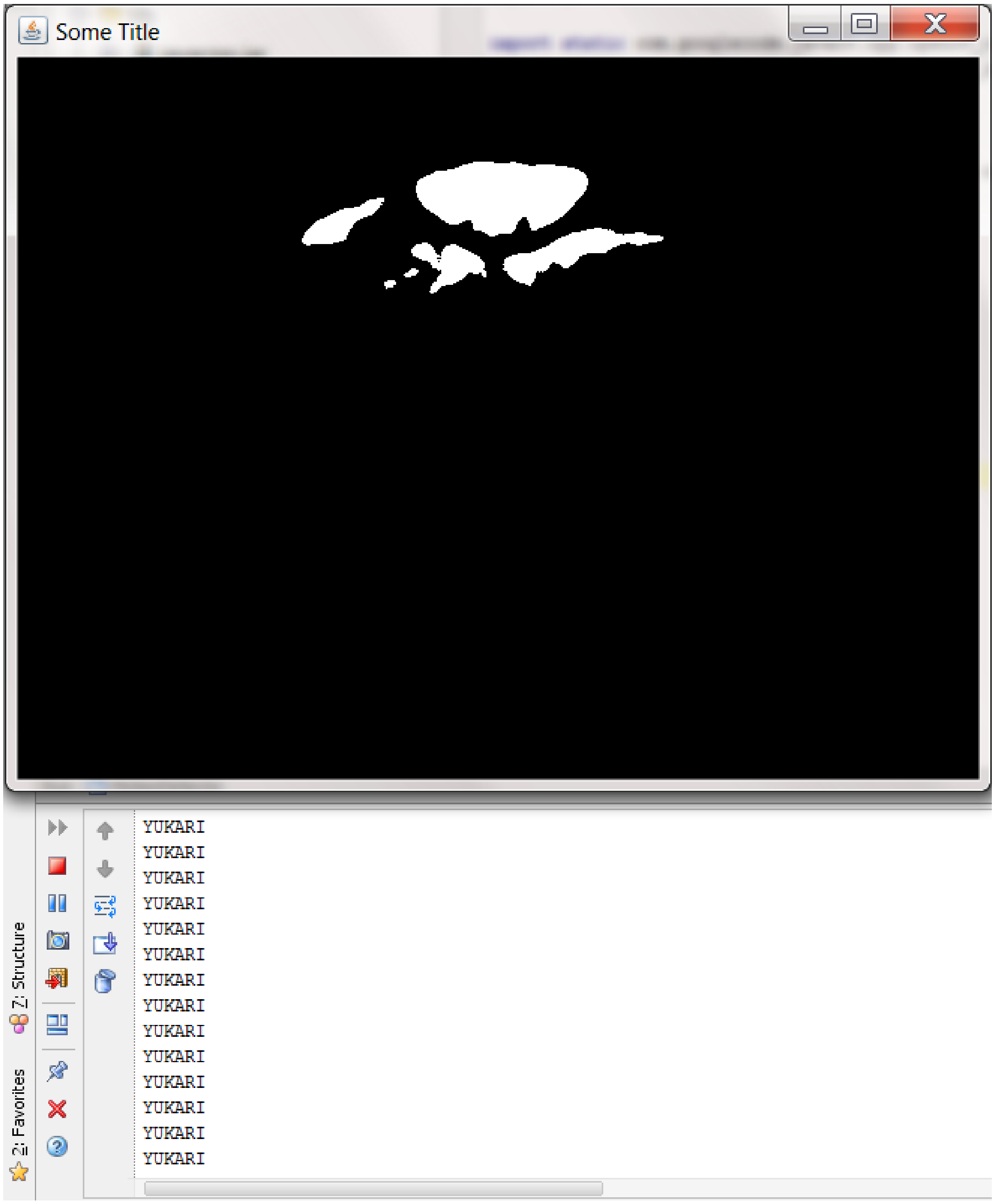
```
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
SAG
```

Figure 4 - The Software Output when User Looks Right



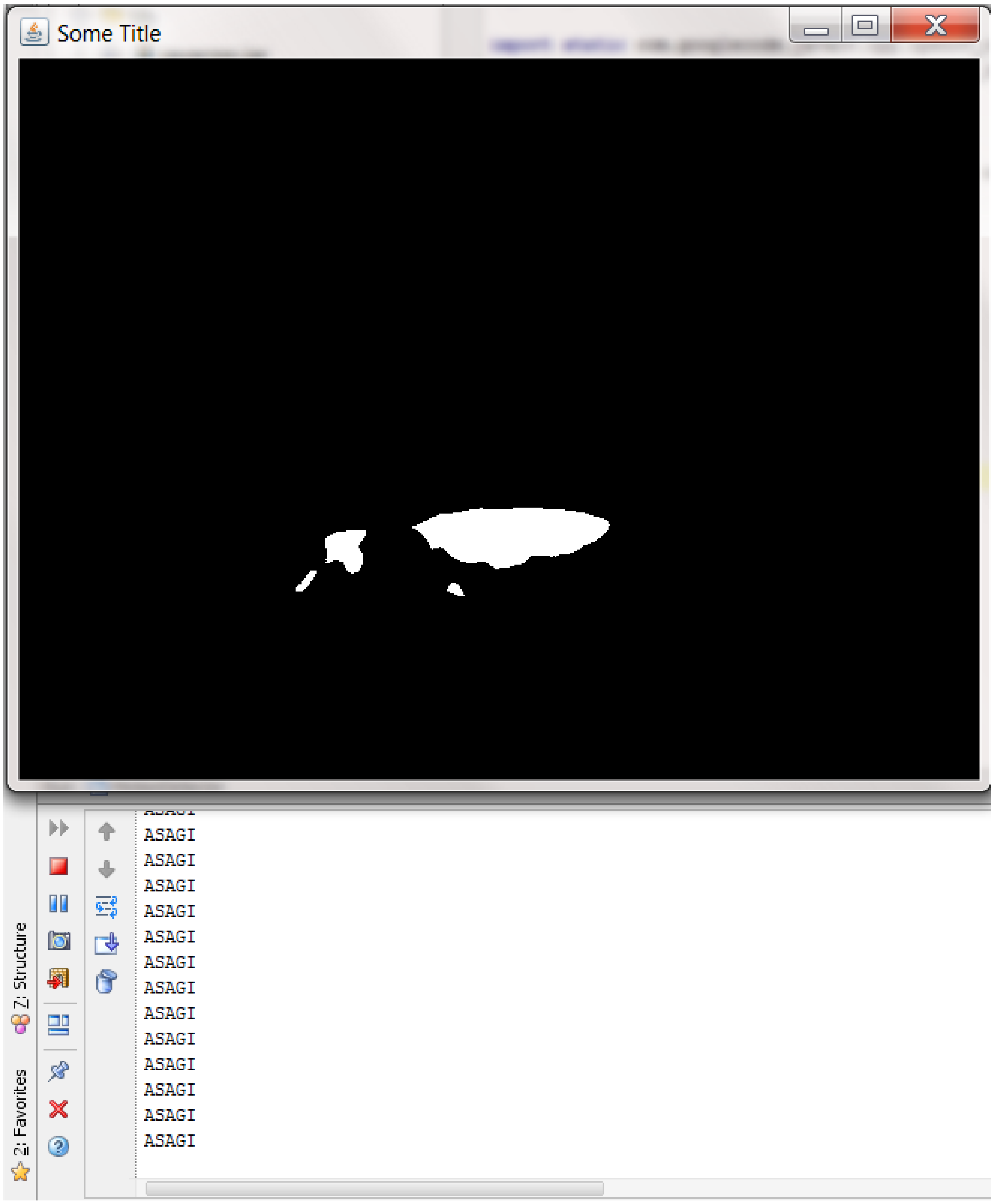
```
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
SOL
```

Figure 5 - The Software Output when User Looks Left



```
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
YUKARI
```

Figure 6 - The Software Output when User Looks Up



```
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
ASAGI
```

Figure 7 - The Software Output when User Looks Down

References

[1] <http://opencv.willowgarage.com/wiki/>

[2] <https://www.gliffy.com/>

[3] <http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html>