

**T.R.**  
**GEBZE TECHNICAL UNIVERSITY**  
**FACULTY OF ENGINEERING**  
**DEPARTMENT OF COMPUTER ENGINEERING**

**SCREEN TRACK AND ACTION**

**EMRE ÖZBAY**

**SUPERVISOR**  
**PROF. FATİH ERDOĞAN SEVİLGİN**

**GEBZE**  
**2022**

**T.R.**  
**GEBZE TECHNICAL UNIVERSITY**  
**FACULTY OF ENGINEERING**  
**COMPUTER ENGINEERING DEPARTMENT**

**SCREEN TRACK AND ACTION**

**EMRE ÖZBAY**

**SUPERVISOR**  
**PROF. FATİH ERDOĞAN SEVİLGİN**

**2022**  
**GEBZE**



GRADUATION PROJECT  
JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 20/01/2022 by the following jury.

**JURY**

Member  
(Supervisor) : Prof. Fatih Erdoğan Sevilgen

Member : Prof. Erchan Aptoula

Member : R.A. Mehmet Burak Koca

# **ABSTRACT**

Screen Track and Action developed for providing bunch of unique automation options to the user by monitoring any selected region of the screen and waiting for predetermined trigger (e.g. detection of user specified text, specific mouse gesture). After detection of the trigger condition program will execute user specified action (e.g. power off computer, extract and copy any text from selected region).

# LIST OF SYMBOLS AND ABBREVIATIONS

## Symbol or

### Abbreviation : Explanation

CNN : Convolutional neural network

OCR : Optical character recognition

QT : QT is a widget toolkit for creating graphical user interfaces

X11 : X Window System is a windowing system for Unix-like OS.

MNIST : Modified National Institute of Standards and Technology database

# CONTENTS

<b>Abstract</b>	<b>iv</b>
<b>List of Symbols and Abbreviations</b>	<b>v</b>
<b>Contents</b>	<b>vi</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>1 GUI Application</b>	<b>1</b>
1.1 Triggers . . . . .	1
1.2 Actions . . . . .	2
1.3 Region Selection . . . . .	2
<b>2 Technical Details</b>	<b>3</b>
2.1 Region Selection . . . . .	3
2.2 Triggers . . . . .	4
2.2.1 Mouse Based Triggers . . . . .	4
2.2.2 Graphical Triggers . . . . .	4
2.2.2.1 Color Detection . . . . .	4
2.2.2.2 Text Detection . . . . .	4
2.2.2.3 Update of Region . . . . .	5
2.2.3 Gesture . . . . .	5
2.3 Actions . . . . .	6
2.3.1 Power Management . . . . .	6
2.3.2 Command Execution . . . . .	6
2.3.3 Text Insertion . . . . .	7
2.3.4 Mouse Based Actions . . . . .	7
2.3.5 Text To Clipboard . . . . .	7
2.3.6 Screenshot of The Region . . . . .	7
<b>3 Test Results</b>	<b>9</b>
<b>4 Conclusions</b>	<b>10</b>
<b>Bibliography</b>	<b>11</b>

# LIST OF FIGURES

1.1	GUI Application . . . . .	1
2.1	Color Picker . . . . .	4
2.2	Text Detection . . . . .	5
2.3	Input Gesture . . . . .	6
2.4	Command Execution . . . . .	6
2.5	Text to Clipboard Example . . . . .	7
2.6	Screenshot Example . . . . .	8

# LIST OF TABLES

3.1	Text detection test results . . . . .	9
3.2	Resource usage test results . . . . .	9



# 1. GUI APPLICATION

Initially there is two dropdown menus and one button. Dropdown menus are for selecting trigger and action. Button triggers region selection part. User selects a trigger and an action from menus, selects the part of the screen to be monitored.

Bottom part of the application updates according to dropdown selections and for getting needed arguments from user (e.g. Text to be searched for in selected region, command to be executed). Only after selection of all the required parts, program will start monitoring region.

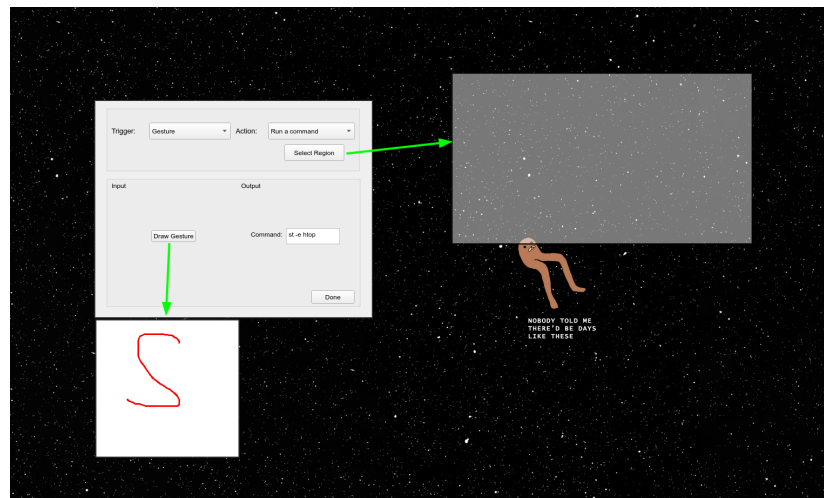


Figure 1.1: GUI Application

## 1.1. Triggers

Trigger is an activity that is decided by the user to be sought by the program. There are 6 triggers for now. Mouse based triggers are platform dependant.

- Gesture
- Color Detection
- Text Detection
- Update of the region
- Mouse Button
- Mouse Position

## 1.2. Actions

Actions are commands to be executed by the program after the trigger detected. Conceptually separated to 10 different actions. The majority of the actions are only works under unix systems.

- Shutdown
- Restart
- Sleep
- Run a command
- Insert text
- Pointer move
- Pointer click
- Get Text to Clipboard
- Screenshot of Region
- Screen Capture Region

## 1.3. Region Selection

After clicking select region button, user can select any rectangular region from their screen. Multiple screen selection is supported as well. After selecting region program is ready to start.

## 2. TECHNICAL DETAILS

### 2.1. Region Selection

After clicking select region button, frameless selection window covers all the screens. This window is transparent and blocks all other active windows. With holding down left mouse button and dragging it a rectangular region can be drawn over selection window. This regions starting point x, y and height, width data saved for later use.

---

**Algorithm 1** Rectangle Drawing

---

```
 $x_{src} \leftarrow 0$ 
 $y_{src} \leftarrow 0$ 
 $x_{end} \leftarrow 0$ 
 $y_{end} \leftarrow 0$ 
 $w \leftarrow 0$ 
 $h \leftarrow 0$ 
while true do
    if buttonPress then                                ▷ Get global X and Y for starting point
         $x_{src} \leftarrow x_{end} \leftarrow \text{current } X$ 
         $y_{src} \leftarrow y_{end} \leftarrow \text{current } Y$ 
    end if
    if dragging then                                    ▷ Update end point
         $x_{end} \leftarrow \text{current } X$ 
         $y_{end} \leftarrow \text{current } Y$ 
        Draw the rectangle on screen
    end if
    if released then                                    ▷ Calculate width and height
         $w \leftarrow x_{end} - x_{src}$ 
         $h \leftarrow y_{end} - y_{src}$ 
        break                                              ▷ Stop the loop
    end if
end while
```

---

## 2.2. Triggers

### 2.2.1. Mouse Based Triggers

For implementing mouse based triggers, global position data and button down status of mouse provided by QT built-in libraries. After selecting region region data will be used for checking if current global cursor position is inside of the user selected region. And if button detection selected which button is pushed inside the region.

### 2.2.2. Graphical Triggers

#### 2.2.2.1. Color Detection

After selection of a color from color picker using gui app, selected color will be searched inside the selected region. Selected regions raw pixel data is feeded into a function for iterating over every pixel and searching for selected color code. Regions raw pixel data is updated every frame.

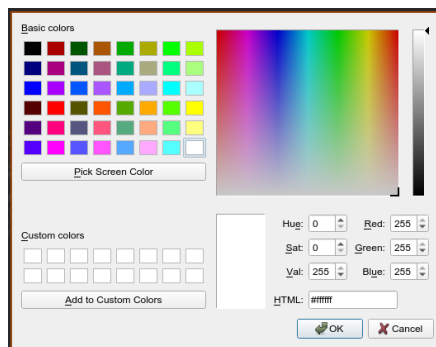


Figure 2.1: Color Picker

#### 2.2.2.2. Text Detection

Tesseract OCR engine[1] used for text detection in region. First step getting regions pixel data and grayscaling it. Tesseract does various image processing operations internally before doing the actual OCR. It generally does a very good job of this, but there will inevitably be cases where it isn't good enough, which can result in a significant reduction in accuracy. Only grayscaling will not give the optimum result so with the help of leptonica library image noise removed.

Since input image is rectangular area on the computer screen there is no need for rotating it. After that, conversion from QImage of QT to PIX data structure of leptonica and applying stated image processing operations will be handled by function which constructs PIX matrix from given QTPixmap matrix. Created PIX matrix will be fed into OCR engine and it will return UTF8 coded text3.1 from image. Given user string will be searched inside output text.

**Example: String to be detected is 'Behind'**

Mild Splendour of the various-vested Night!  
Mother of wildly-working visions! hail!  
I watch thy gliding, while with watery light  
Thy weak eye glimmers through a fleecy veil;  
And when thou lovest thy pale orb to shroud  
**Behind** the gather'd blackness lost on high;  
And when thou dartest from the wind-rent cloud  
Thy placid lightning o'er the awaken'd sky.

Figure 2.2: Text Detection

### 2.2.2.3. Update of Region

Regions previous frame is stored inside the structure and it will be updated with constant frequency which is refresh rate of the display. Using internal QT image operations previous frame and captured current frame is compared. If there is any difference between those frames action trigger signal will be sent.

### 2.2.3. Gesture

Gestures are movements that you make with you mouse (or your pen, finger etc.) while holding down a specific mouse button. Gesture detection was one of the most complex triggers to implement. At first I tried to create my own dataset for some possible gestures but it took so long to create data for just one gesture type which didn't meet the accuracy goal of my project. So I opt-out from creating my own CNN model. Extended MNIST letter dataset[3] used for training CNN model which means user could draw any character from alphabet and program will be recognize it.

CNN model trained tested with python using tensorflow and keras. Tensorflow C++ library was too large and was consuming so much resource for simple model like this, so model converted to special file type using frugally-deep[2].

Recognition will executed 2 times. First for setting a gesture and second to actually recognize given input. When user draws their gesture at the beginning it is saved as a QImage. After some image operations similar to ones that are done in OCR parts 2.2.2.2 output image scaled to input size of model. Model will return prediction and it will be stored inside object for later comparison between input gesture. The input gesture to be compared will be provided by user drawing it inside the selected region using their mouse.

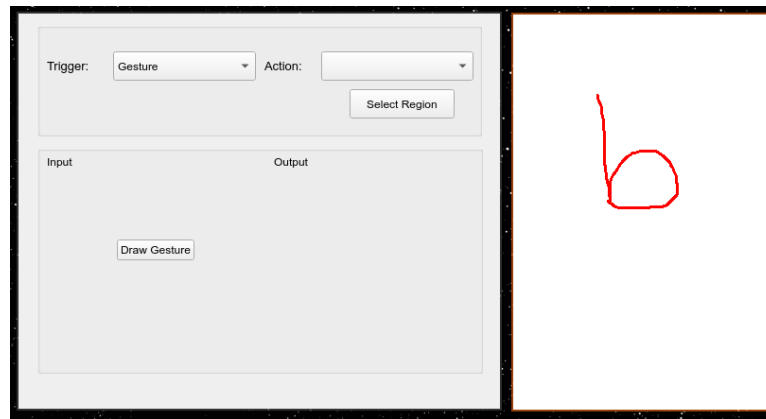


Figure 2.3: Input Gesture

## 2.3. Actions

### 2.3.1. Power Management

Shutdown, Sleep and Restart signals will be sent to operating system using QT process management system.

### 2.3.2. Command Execution

Any valid shell command can be executed using this part. Processes will be started detached so if user closes main program, started process will be stay unaffected.

#### Example: Command

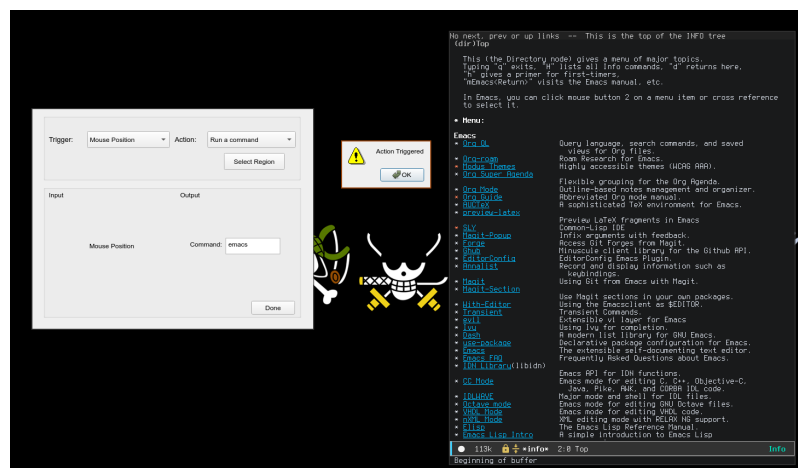


Figure 2.4: Command Execution

### 2.3.3. Text Insertion

Text (which is given by user) will be inserted to currently active window. Xdo library used for getting currently active window and inserting given text. There is little delay between trigger and text insertion because window manager of the system needs a little time to return active window and xdo function will insert text character by character.

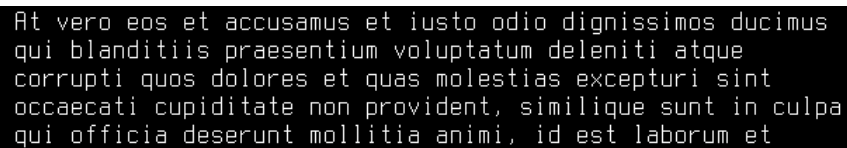
### 2.3.4. Mouse Based Actions

Mouse position to move and/or button to click will be given from the user by using gui app. Xdo library used for moving mouse to given location and simulating button click event. QT can't set cursor position and simulate button clicking event globally because of widget based approach. So platform specific (X11[4]) code used for simulating these events.

### 2.3.5. Text To Clipboard

All the image manipulations and conversion operations stated in 2.2.2.2 are done in this part again. After feeding image into OCR engine output3.1 text copied into system-wide clipboard by custom shell script and executed with text as parameter.

#### Example:



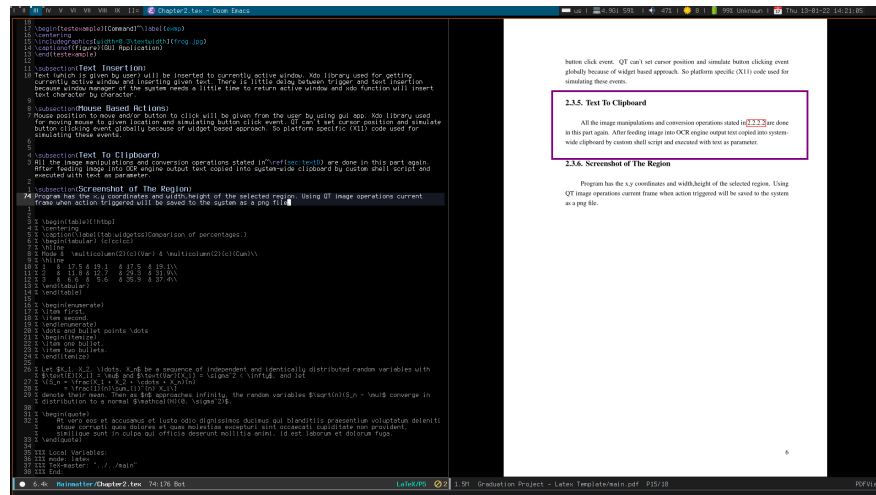
```
At vero eos et accusamus et iusto odio dignissimos ducimus  
qui blanditiis praesentium voluptatum deleniti atque  
corrupti quos dolores et quas molestias excepturi sint  
occaecati cupiditate non provident, similique sunt in culpa  
qui officia deserunt mollitia animi, id est laborum et
```

Figure 2.5: Text to Clipboard Example

*“ At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum  
deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non  
provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.  
”*

### 2.3.6. Screenshot of The Region

Program has the x,y coordinates and width,height of the selected region. Using QT image operations current frame when action triggered will be saved to the system as a png file.



(a) Current screen with selected region

## 2.3.5. Text To Clipboard

All the image manipulations and conversion operations stated in [2.2.2.2](#) are done in this part again. After feeding image into OCR engine output text copied into system-wide clipboard by custom shell script and executed with text as parameter.

(b) Saved screenshot

Figure 2.6: Screenshot Example



### 3. TEST RESULTS

Table 3.1: Text detection test results

Mode	# of Tests	>%90 accurate	Total Accuracy
Dark background and light text	100	94	%94
Light background and dark text	100	98	%98

This data collected by using a script for taking screenshots of different part of screen and feeding into OCR engine. Text accuracy calculated using Levenshtein Distance Computing Algorithm[5].

Table 3.2: Resource usage test results

Process Type	CPU Usage	RAM Usage
Text detection and copy to clipboard	%17	89.20MB
Cursor detection and copy text to clipboard	%3	12.60MB
Average of all	~%9	~28MB

First process is the most resource hungry trigger-action combination. Data collected using unix process manager htop. Each trigger-action combination tested with various selected region sizes <sup>1</sup>at least 10 times and average data used in table.

---

<sup>1</sup>Size of the region effects cpu and ram usage significantly. Region size and need for resource to store and process are proportional.

## 4. CONCLUSIONS

With this project end user has the ability to create unique automation rules for their PC environment. Specifically gesture detection part of the project was designed with Tablet PCs and touch screen laptops in mind and can be used very easily without access to a keyboard.

In addition to the gesture recognition, OCR based trigger and actions were very resource hungry processes<sup>3.2</sup> but after long hours of optimization and configuration, program now meets the success criterias decided at the beginning of the project.

While working on this project I learned a lot about crossplatform desktop application development and gained experience on the fields like computer vision, image processing, basics of unix system programming and deep-learning.

# BIBLIOGRAPHY

- [1] *Tesseract open source ocr engine*, <https://github.com/tesseract-ocr/tesseract>.
- [2] *Header-only library for using keras (tensorflow) models in c++*. <https://github.com/Dobiasd/frugally-deep>.
- [3] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, *Emnist: An extension of mnist to handwritten letters*, 2017. arXiv: 1702.05373 [cs.CV].
- [4] *X11 keyboard and pointer events*, <https://tronche.com/gui/x/xlib/events/keyboard-pointer/keyboard-pointer.html>.
- [5] P. E. Black. "Levenshtein distance." (1999), [Online]. Available: <https://www.nist.gov/dads/HTML/Levenshtein.html>.