Passive RFID Gesture Recognition
Final Report

Team Members:
Christopher Chen
Justin Cruz
Sai Kotikalapudi
Kyung-Tack (Terry) Oh

Project Advisor:
Ivan Marsic

Guidance from:
Young Lee
Xinyu Li
Table of Contents
Abstract .................................................................................................................. 3
Introduction ............................................................................................................. 3
Goals ....................................................................................................................... 4
System Overview .................................................................................................... 4
  Proposed System Architecture ........................................................................... 4
  Demonstrated System Architecture ................................................................. 5
RFID Antenna .......................................................................................................... 6
RFID Reader ........................................................................................................... 7
RFID Tag ................................................................................................................ 8
Router ..................................................................................................................... 8
Proposed Interface Design ................................................................................... 9
Demonstrated Interface Design ........................................................................... 10
Proposed Data Analysis Algorithms .................................................................. 11
Proposed Recognition Algorithms ...................................................................... 12
Demonstrated Recognition Algorithms ............................................................... 14
Problems and Challenges ..................................................................................... 16
Analysis ................................................................................................................ 18
  RFID Analysis ..................................................................................................... 18
  Recognition Algorithms ...................................................................................... 20
System Implementation ......................................................................................... 23
  Implementation – Data Live Streaming ............................................................. 23
  Implementation – Live Stream Plot .................................................................. 23
  Implementation – Data Communication (Java to Matlab) ............................... 24
  Implementation - Feature Extraction ................................................................. 25
  Implementation – Music Player ......................................................................... 26
Future Consideration ............................................................................................ 28
  Sample Product Design .................................................................................... 28
  Alternate Forms of Data Analysis .................................................................... 28
Cost Estimates and Analysis ................................................................................. 28
Individual Contributions.................................................................................................................................................. 29
Appendix I: Project Code.............................................................................................................................................. 30
  Data Streaming from RFID Readers ......................................................................................................................... 30
  Live Plot of Antenna Data .......................................................................................................................................... 30
  Gesture Recognition Algorithms............................................................................................................................. 31
  Hog Feature Extraction ............................................................................................................................................. 39
  Data Communication (Java to Matlab) .................................................................................................................... 41
  Music Player User Interface ................................................................................................................................... 42
  Normalization functions (Crop, Rescale, black/white) ............................................................................................. 49
  Algorithm Test GUI .................................................................................................................................................. 50
References ................................................................................................................................................................. 62
Abstract

The Purpose of this project was to explore the capabilities of passive Radio Frequency Identification (RFID) tags in the field of gesture recognition and introduce a designed and interface that allows recognition of gestures when using RFID tags. Through experimentation, we designed a frontal interface for getting information from RFID tags and carefully selected an analytical algorithm to recognize the gesture made. In order to interface with the system, we have readers that take input data of the tags as a gesture is being made, most significant of which are, Radio Signal Strength Indication (RSSI) and the time stamp of the reading. Using these readings, we are able to manipulate and analyze them using our data analysis algorithms to recognize the gesture and classify it. This idea can potentially introduce a new form of gesture recognition. We demonstrate its use by showing gesture based interaction of a music player (Please see Implantation – Music Player for more detail).

Introduction

Technology, today, has become the focus of human advancement. One of the technologies that has come into today’s spotlight is the field of gesture recognition. Although it started as an alternative interface with computers [1], gesture recognition is now used for wider ranges of interaction and analytics. From immersive game technology like the Xbox Kinect [2], to the analysis of human emotions and behavior with the use of facial gestures and body movements [3], we find gesture recognition in almost every aspect of life. Even though there are so many diverse implementations in gesture recognition, there also has been little, to none, research done on many of the alternatives inputs for gestures, one being passive RFID tags. What is a RFID Tag? Radio Frequency Identification Tags, also known as RFID tags are wireless tags use of radio-frequency electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects. What is a Passive RFID tag? Passive RFID tags are cheap RFID tags that do not require any internal batteries but relies on the RFID reader as a power source [4]. What is the point of having this alternative to current Gesture recognition technology? Major gesture recognition technologies today include: Wired gloves, depth-aware or stereo camera, and controller-based gestures. All of these technologies have both flaws as well as advantages. In case of wired gloves, they are bulky equipment’s that not only require heavy use of batteries but also need heavy and accurate gyroscopes while also generally being expensive and requiring high maintenance. When using depth-aware or stereo cameras, the recognition requires environmental perfection like lighting and usually expensive for example the Kinect controller for Xbox which is currently selling for $100 (without including the Xbox). As stated, most of these technologies are usually expensive and generally have a high dependence on power. Fortunately, using passive RFID tags can alleviate these problems. Passive RFID tags themselves provide high source of power efficiency, cheap, light, and low maintenance. Our idea is to leverage passive RFID’s advantages, to create a new tool for gesture recognition. However, there isn’t enough research to determine the feasibility of RFID as a source for gesture recognition.
Goals

The Goals were split into 2 distinct but important tasks:

1. Analyze the possibilities and limitations of the RFID technology. Please look at the Analysis section for detailed analysis of RFID’s capabilities.

2. Demonstrate the practical use of the RFID as a source of gesture recognition. Below we describe the design to demonstrate how RFID tags can be used for recognition. The demonstration is done through the use of a music player that maps gestures to some music player features.

System Overview

Proposed System Architecture

The figure below demonstrates the system architecture for the gesture interface we proposed for the project. Our gesture interface consists of a RFID tag, three Radio Frequency Antennas, a Radio Frequency Reader, and a Router. The three RF antennas are connected to the reader, which processes and synchronizes the data of the RFID tag read by the reader, through timestamps. It then feeds the data collected by these antennas over Ethernet to the router which packages and sends it wirelessly or through Ethernet to an external computer for displaying and recognizing the data into the gesture done by the tag.

![Diagram of system architecture]

Figure 1: Proposed System Architecture of the Gesture Interface
Demonstrated System Architecture

The figure below demonstrates the system architecture for the gesture interface used in this project. Our gesture interface consists of a RFID tag, two Radio Frequency Antennas, two Radio Frequency Readers, and a Router. The two RF antennas are each connected to an individual reader, which processes and synchronizes the antenna’s data individually, through timestamps. It then feeds the data collected by these antennas over Ethernet to the router which transmits it wirelessly or through Ethernet to an external computer for displaying and recognizing the data into a gesture.

Figure 2: Demonstrated System Architecture of the Gesture Interface
RFID Antenna
Alien 915 Mhz Circular UHF Antenna (ALR-9610-AL)

The Alien 915 MHz Circular Antenna uses circular polarization to distribute the Ultra High Frequency (UHF) energy uniformly in a radially symmetrical pattern, providing the ability to read RFID tags regardless of orientation. The ALR-9610 series of UHF antennas are specially designed to deliver optimum performance when used with Alien RFID readers and optimized for the 902 MHz to 928 MHz ISM frequency band.[5] The design methodology achieves maximum efficiency and performance across the provided frequency band and tag orientations. The Voltage Standing Wave Ratio (VSWR) and axial ratios are both ideal and allow the user to achieve the optimal performance for this type of antenna. The antenna provides beam width (3DB) of 40 degrees.[5] Throughout the process of the project, we utilized from a single antenna for data reception to multiple antennas for location awareness.

Figure 3: RFID Antenna
RFID Reader
Alien Enterprise RFID Reader (ALR-9900)

The Alien Enterprise Category Reader enables users to deploy manageable, robust, EPC Gen 2 RFID solutions for supply chain, manufacturing and asset management applications.[6] This reader delivers ideal performance with optimal receive sensitivity, enhanced interference rejection, Dynamic authentication, a monostatic antenna architecture and a compact footprint, delivering high read rates for demanding applications. The reader is supported with well-documented SDK with compatibility with .NET, Ruby, and Java libraries to create a custom interfaces to control the reader. The reader also offers several methods for interference mitigation that provides that provide a powerful solution to the challenge of noisy environments. The reader facilitates a low duty cycle “sniff & read”[6] mode for applications where motion detections is not an option, or where power conservation or RF interference is an issue.

With the provided RFID reader combined with the tasks given for the project, utilizing the features of the system was necessary to achieve the proposed ideal requirements. High rate of data transmission is required for a real time gesture recognition with minimal delay.

Figure 4: RFID Reader
**RFID Tag**
Alien Squiggle Inlay (ALN-9640)

Radio-frequency identification (RFID) tags are the centerpiece of any RFID system. RFID tags are offered in different sizes, frequencies and can be affixed to a variety of surfaces. The tags store item data essential to any RFID-based tracking system. The Squiggle Inlay tag used contains Higgs 3 UHF RFID IC[7] that uses the Squiggle antenna design for reliability and efficiency at competitive costs. With the pre-programmed unique, unalterable 64-bit serial number, it is possible to identify multiple tags concurrently without errors.

![RFID Tag](image)

**Figure 5: RFID Tag**

**Router**
Linksys WRT54G Wireless-G Broadband Router

The role of the router in our project was to communicate between the RFID reader and the PC. Data generated from the RFID reader is transferred to the data recognition program over TCP/IP networking.

![Router](image)

**Figure 6: Router**
Proposed Interface Design

The figure below demonstrates the system architecture for the gesture interface design we proposed for the project. Our design consists of using multiple Antennas as a source for localization of the position of a Passive RFID tag in space. The lines in the figure below represent the scope of the region where the tag can be read by reader. This is based on previous research done in the Gesture recognition using RFID technology paper[8].

Figure 7: Proposed Interface Design
Demonstrated Interface Design

For the demonstration we used the idea of zoning for the interface design. Our initial zoning idea was to create zones in space using multiple RFID readers. These zones would localize the tag allowing a pattern to be recognized by the zones in which it crosses into. Each zone is distinguished by what RFID reader is capable of reading the tag.[8]

![Figure 8: Zoning Concept](image)

For example: a tag in section 5 in Figure 1 will be read by all three readers but when it moves to section 2, it will only be read by the red and blue readers. So when we implemented this orientation to the three gestures, the L gesture will cross into (zones 3, 2, 1, 6, then 7), V gesture will cross into (zones 1, 2, 5, 6, then 7), N gesture will cross into (zones 1,2,3, 2,7, then 6). As a result of gestures having a distinct sequence and combination of zones they would enter, there is great potential in the number of recognizable gestures by adding more antennas. More antennas would increase the resolution of the zones and allow for greater complexity in the gestures.

Due to the time constraints and complications about the obstructive nature of using wide angle antennas, we decided to initially start with a simple two antenna orientation as a foundation for 3, 4, 5, ... antenna orientations.

![Figure 9: RFID Antenna Experimental Orientation for Zoning](image)
Proposed Data Analysis Algorithms

In order to smooth out the data, we used a moving average filter which had promising results on the stationary data but proved quite ineffective when applied to data correlating to gestures. This common digital signal processing filter is ideal for signal where you want the zero frequency response. When we did the frequency response of a stationary data set, we noticed the following plot.

![Frequency Response of RSSI](image1)

**Figure 10: Frequency Response of RSSI**

From the figure 11 below [10] we can see how a moving average filter acts as a low pass filter allowing signals at a frequency of zero to remain while attenuating the values for all other frequencies.

![Frequency Response of Moving Average Filter](image2)

**Figure 11: Frequency Response of Moving Average Filter**
Proposed Recognition Algorithms

Our initial implementation of image recognition used pixel matching to recognize images provided by a visualization of a gesture and matching it to a known gesture movement in our library.

![GUI for Pixel Matching Algorithm](image)

Figure 12: GUI for Pixel Matching Algorithm

We initially defined three gestures:

![C-Shape, L-Shape, V-Shape](image)

Figure 13: C-Shape, L-Shape, V-Shape (from left to right)

Using these images as a base reference in our library, we gave our program different inputs to see whether or not it would be able to recognize what gesture was recorded.
With several trial and error, we noticed that using pixel matching wouldn’t work without a proper normalization of the image. We then worked on cropping and re-scaling the image to improve our results. We implemented code that would set the boundary of the given image to the first significant points on all sides. We also, re-scaled the image to match the known image size in our library.
Our next implementation involved an aspect of scaling both axis. This implementation takes a look at both the x-axis and y-axis and plots the features of the image as you move across. For example:

![Image of GUI for Scaling Algorithm](image)

**Figure 16: GUI for Scaling Algorithm (x-axis = pixel row/column number and y-axis = pixel count)**

The left image is a visualization of scaling on the x-axis and the right image is a visualization of scaling on the y-axis. The dark blue line is scaling for the given input image of a V-shape, the thin blue line is the scaling feature of a C-Shape, the thin black line is the scaling feature of the V-shape, and the thin red line is the scaling feature of the L-Shape. Through the use of MATLAB’s regression function we were able to compare which line was the best fit. Even though this implementation had its flaws, we decided to move to a more advanced algorithm.

**Demonstrated Recognition Algorithms**

Our final implementation involved using histogram of oriented gradients. This algorithm splits the given image into even sections that overlap and counts the occurrences of oriented gradients across the sections. We took the output of the reader, created a visualization of the data, and put it into our MATLAB code that implemented Histogram of Oriented Gradients (HOG). This algorithm takes a histogram of gradients and tries to find the direction or orientation of the pixel giving it a vector. Then, it compares the features to the pre-defined gestures in our library and matches the data to the gesture that matches the closest. I tried to compare the features using two methods. The first method takes the
occurrences of gradients of the input image and subtracts them to the occurrences of gradients of the images in the library. If the pictures match then their differences would be close to zero. I take the mean of the vector of occurrences of gradients and the one closest to zero is the match. The second method of comparison takes the covariance of the two features and the image in the library with the highest covariance compared to the input image is the image of best fit.

![Graphs showing HOG features and difference in L² norm](image.png)

Figure 17: GUI for HOG Algorithm

Through the use of this knowledge for one sensor, we were able to scale upwards to multiple sensors and create gestures that we would be able to work with. These features allow us to create a defined gesture using these features and correspond the gesture to an action. Our idea was to experiment using this gesture recognition technique to control music player.
Problems and Challenges

Our initial proposal was to localize the RFID using the RF antennas based off the received signal strength indicator. However, due to the multipath fading effect [11] which causes the fluctuating nature of the RSSI, we deemed triangulation not possible or remotely ideal. Judging from the figure below, we saw that a single RSSI value could correlate to multiple distance locations which would cause inaccurate triangulation results.

![Plot of Stationary Data](image)

**Figure 18: Effects of Multipath Fading**

In response to the fluctuating RSSI values, we tried implementing a moving average filter in order to smooth the RSSI values and obtain more reliable metrics. We knew we would be unable to filter out the multipath fading effect but thought it would enable us to visualize the general trend of the data as the passive RFID moved. When our filter was implemented on stationary data, the general trend was visible and clear. However, when tested using streaming of a single antenna, we realized that the filter was unable to keep up with the movements of a person and the data got attenuated. In order to improve the speed of the filter to visualize the data, we tried increasing the error but overshooting occurred and the data changed before it could stabilize. This led us to reconsider our initial proposal and implement a zone method instead.

Implementing a zone method required real time streaming of multi antennas [8]. When we initially designed our zone orientation, we planned for a three antenna setup based off of a narrow beam width and clear field of detections. However, through our preliminary testing, we discovered that the specific antennas we were using had a wide field of detection which grew proportionally as distance increased. This made the feasibility of our zoning idea to be rather impractical with our current equipment. In order to implement it, the location of antennas would be fixed a foot or two above the
RFID. As a result, we downsized to a two antenna orientation and made the antennas face the user so zone size would be within a designated preset range.

As we were pursuing multi-antenna real time streaming, we realized the limitations of the readers we were using. There was a one second delay switching time from one antenna to another. As a result, we were forced to change our gesture interface to pair one reader per antenna increasing the overall bulkiness and increasing the overall calibration time in new settings since the RSSI fluctuations are affected by environment. [11] It also led to complications in synchronization and live streaming.

Synchronization was difficult to implement because the hardware clock on one of the readers was skewed and did not give synchronized timestamps. So we needed to adjust the skew of one of the readers in order to be able to get a better plot of the data relative to each other.

One other problem that we faced was trying to make our program continually listen for gestures and implement an action, aka livestream data collection. Trying to recognize when a gesture started was easily determined by us visually but ended up being very hard to implement using mathematical models. In the end, we were able to make pseudo-livestream data collection where we controlled a button that initiated a program to listen for gestures and click the button again when the gesture was complete. This allowed for the user to control when he or she wanted a gesture to be recognized without accidentally doing something unwanted.
Analysis

RFID Analysis

Here are the results of our RFID data as we progressed from the beginning of the semester until the end.

Figure 19: Plots of Moving Towards and Side to Side

As we previously mentioned, the Multipath Fading effect is due to the waves traveling across different paths of varying lengths as a result of reflecting, diffracting, and scattering off objects in the environment [11].
From this plot we can see that given enough time, the filter we have would successfully stabilize. This demonstrates the general trend of the RSSI signal as we intended. However, gesture movements are never that slow. As a result, we received plots such as the one below.
Recognition Algorithms

Here are the results of our gesture recognition algorithms as we progressed from the beginning of the semester till the end.

Figure 22: Pixel Recognition - Result Matched to “L-Shape”

Figure 23: Scaling Recognition - Result Matched to “L-Shape”

Figure 24: HOG Implementation - Result Matched to “L-Shape”
Given the input on the left side, this is our results:

<table>
<thead>
<tr>
<th>Pixel Matching</th>
<th>Scaling</th>
<th>HOG (Comparing Mean)</th>
<th>HOG (Comparing Covariance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.4 %</td>
<td>99.6%</td>
<td>1 - 0.851 x 10^(-2) = 99.1%</td>
<td>cov = 0.0238</td>
</tr>
</tbody>
</table>

**Figure 25**: Pixel Recognition - Result Matched to “L-Shape”

**Figure 26**: Scaling Recognition - Result Matched to “L-Shape”
Figure 27: HOG Implementation - Result Matched to "V-Shape"

Given the input on the left side, this is our results:

<table>
<thead>
<tr>
<th>Pixel Matching</th>
<th>Scaling</th>
<th>HOG (Comparing Mean)</th>
<th>HOG (Comparing Covariance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.4 %</td>
<td>95.5%</td>
<td>1 - 0.118 x 10^{-1} = 98.8%</td>
<td>cov = 0.0538</td>
</tr>
</tbody>
</table>
System Implementation

Implementation – Data Live Streaming

The RFID reader has been setup to continuously transmit data to a specified IP and Port (Look as Appendix I for the code on Data Live Stream), this data is transmitted at 6 data points per second from each reader. The reader processes the raw data it receives about all tags in the zone of the Antenna and then aggregates this data. This data is made available to transfer when either streaming or requesting and I/O for antenna data. The code has been set up to only accept a specified tag and only stream three sets of data. The three sets of data include the antenna number, the RSSI and the timestamp. The above was also done for the other reader as well in order to create 2 sources of information for a specified RFID tag. Figure 28 is an example of the streamed data received from an antenna.

![Sample Data for a specified RFID Tag](image.png)

Implementation – Live Stream Plot

In order to visualize the data from the reader, and live plot was made of the data received from streaming. This plot includes data from both reader sources, and makes a plot of Timestamp VS RSSI from the data received from the Data Live Stream program. (Please see Appendix I Live Plot of Antenna Data for the related code) Figure () shows a sample plot resulting from the program.
Implementation – Data Communication (Java to Matlab)

In order to make a live version of gesture recognition, we needed a link to transfer live data from Java which was used to receive data to Matlab which housed the recognition algorithm code. So, a data communicator was made that transmits data over TCP/IP connection locally between Matlab and Java. (Please see Appendix I Data Communication for the related code)

Figure 29: Sample Plot for a specified RFID Tag
Implementation - Feature Extraction

Features of our passive RFID tag:

<table>
<thead>
<tr>
<th>Case 1: Approaching the Sensor</th>
<th>Case 2: Leaving the Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Graph for Case 1" /></td>
<td><img src="image2" alt="Graph for Case 2" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 3: Approaching the Sensor then Leaving</th>
<th>Case 4: Leaving the Sensor then Approaching</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Graph for Case 3" /></td>
<td><img src="image4" alt="Graph for Case 4" /></td>
</tr>
</tbody>
</table>
## Implementation – Music Player

Here definitions of the gestures for controlling our music player:

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Image</th>
<th>Feature for Antenna 2</th>
<th>Feature for Antenna 3</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antenna 2 to 3</td>
<td><img src="image1.png" alt="Image" /></td>
<td>Case 2</td>
<td>Case 1</td>
<td>Skip Song</td>
</tr>
<tr>
<td>Antenna 3 to 2</td>
<td><img src="image2.png" alt="Image" /></td>
<td>Case 1</td>
<td>Case 2</td>
<td>Previous Song</td>
</tr>
<tr>
<td>Antenna 2 to 3 to 2</td>
<td><img src="image3.png" alt="Image" /></td>
<td>Case 4</td>
<td>Case 3</td>
<td>Fast Forward</td>
</tr>
<tr>
<td>Antenna 3 to 2 to 3</td>
<td><img src="image4.png" alt="Image" /></td>
<td>Case 3</td>
<td>Case 4</td>
<td>Pause</td>
</tr>
</tbody>
</table>
We created a simple music player in order to demonstrate the application of gesture recognition to control a real world object. Successful control of a music player would demonstrate the feasibility of gesture recognition using passive RFID tags. As can be seen from our YouTube video and demo, we were able to control our music player using the gestures above using pseudo-livestream data collection where we controlled a button that initiated a program to listen for gestures and click the button again when the gesture was complete.

![Music Player](image)

**Figure 30: Music Player**

Using these features allows us to define multiple gestures and expand the library of gestures by using different combinations. We can further expand these gestures by using multiple antennas, allowing for more combinations of gestures. Through experimentation using these RFID readers, we have come to the conclusion that these antennas are not the best use for future trials because these antennas were not made for gesture recognition. Using these antennas gave us a large range for a zone allowing for a few large gestures and discouraging smaller ones due to the resolution it provides. With narrower antennas, additional gestures will be plausible without having to be so large or exaggerated.
Future Consideration

Sample Product Design

In order to make a feasible design for indoor use as a gesture recognition interface, a sample design was made for production. This sample design however requires the making of proprietary equipment. Figure (1) is a picture of the design. The idea is to expand the idea of zoning and feature extraction and HOG on a smaller scale of readers that are more accurate in being able to recognize the smallest of features.

Alternate Forms of Data Analysis

Visualizing the general trend of the real time streaming data by smoothing the RSSI values, would be more computationally efficient and practical with the use of an IIR filter. IIR filters are typically used in real time data analyses due to minimizing time delay and achieving long impulse responses.[9]

Cost Estimates and Analysis

Sensing Equipment:
1. Alien Squiggle Inlay (ALN-9640) ~ 50 cents(sold in bulk of 500 for $250)
2. 2 Alien Antennas ~ $70/per unit
3. 2 Alien ALR-9900+ Enterprise RFID Reader ~ 1,774/per unit

Processing System:
1. Computer
2. Router ~ $50

Estimated Total Cost: $3,738.00 using demonstration setup

This cost estimate is not at all the actually cost for the project since we spent a total of approximately 20 dollars to make this project. All of this equipment was already obtained and loaned to us by Professor Marsic. However, if we had to obtain the equipment we had to use the feasibility of this project is quite low. We would definitely want to obtain or create cheaper antennas with narrower beam widths and cheaper readers. Currently the narrowest beam width we could find is 50 degrees. Due to the typical nature RF antennas are used for, creating one might be more promising. With more research in narrower antennas and interest in RFID gesture recognition, the cost of a system would decrease and eventually the system will be more practical.
Individual Contributions

Christopher Chen
- Demonstrated Interface Design
- Implementation – Data Communication
- Implementation – Music Player

Justin Cruz
- Proposed and Demonstrated Recognition Algorithms
- Implementation – Data Analysis (HOG)
- Implementation – Feature Definition

Sai Kotikalapudi
- Proposed Interface Design
- Implementation – Data Live Stream
- Implementation – Live Stream Plot

Kyung-Tack (Terry) Oh
- Proposed and Demonstrated Recognition Algorithms
- Implementation – Data Analysis (HOG)
- Implementation – Feature Definition
Appendix I: Project Code

Data Streaming from RFID Readers

```java
package SetUpStream;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Demo {
    public static final void main(String args[]){
        try {
            TagStream2 t2 = new TagStream2();
            TagStream3 t3 = new TagStream3();

            while(true)
            {
                System.out.println("To start reading gesture press Enter.");
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                br.readLine();
                t2.service.startService();
                t3.service.startService();
                System.out.println("To stop reading gesture press Enter.");
                br.readLine();
                t2.service.stopService();
                t3.service.stopService();
                t2.write();
                t3.write();
            }
        } catch (Exception e) {
                System.out.println("Error:" + e.toString());
            }
    }
}
```

Live Plot of Antenna Data

```java
package PlotStream;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.swing.JFrame;
import org.math.plot.Plot2DPanel;
import javax.swing.JFrame;
```
public class Demo {
    public static final void main(String args[]) {
        try {
            Plot2DPanel plot = new Plot2DPanel();
            JFrame frame = new JFrame("RSSI vs Timestamp Plot");
            frame.setSize(600, 600);
            frame.setContentPane(plot);
            frame.setVisible(true);
            TagStream2Plot t2 = new TagStream2Plot(plot);
            TagStream3Plot t3 = new TagStream3Plot(plot);

            while (true) {
                frame.setContentPane(plot);
                frame.setVisible(true);
                System.out.println("To start reading gesture press Enter.");
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                br.readLine();
                plot.removeAllPlots();
                t2.service.startService();
                t3.service.startService();
                System.out.println("To stop reading gesture press Enter.");
                br.readLine();
                t2.service.stopService();
                t3.service.stopService();

            }
        } catch (Exception e) {
            System.out.println("Error:" + e.toString());
        }
    }
}

Gesture Recognition Algorithms

function v = finaldemo(filename1, filename2)
% Remember to change the folders.
%% Initialize Library:
% sts2 = imread('patterns/lib/signals/ant2/2to3feature.png');
% sts2_2 = imread('patterns/lib/signals/ant2/sideToSide2.png');
% stsandb2 = imread('patterns/lib/signals/ant2/2to3to2feature.png');
% stsandb2_2 = imread('patterns/lib/signals/ant2/2to3to2feature3.png');
% stsandb2_3 = imread('patterns/lib/signals/ant2/sideToSideandBack2.png');
% stsandb2_4 = imread('patterns/lib/signals/ant2/beforedoomsday2.png');
% stsandb2_5 = imread('patterns/lib/signals/ant2/beforedoomsday2_2.png');
% stsandb2_6 = imread('patterns/lib/signals/ant2/beforedoomsday2_3.png');
% st3 = imread('patterns/lib/signals/ant3/sideToSide3.png');
% stsandb3 = imread('patterns/lib/signals/ant3/sideToSideandBack3.png');

enter = imread('patterns/lib/projectday2/entering.png');
enter2 = imread('patterns/lib/projectday2/entering2.png');
leaving = imread('patterns/lib/projectday2/leaving.png');
leaving2 = imread('patterns/lib/projectday2/leaving2.png');
enterandexit = imread('patterns/lib/projectday2/enterandexit.png');
enterandexit2 = imread('patterns/lib/projectday2/enterandexit2.png');
exitan

terandenter = imread('patterns/lib/projectday2/exitandenter.png');
exitan
terandenter2 = imread('patterns/lib/projectday2/exitandenter2.png');

xscale_new = 640; yscale_new = 640; % Set xscale and yscale

% Gray Scale Images
enter_g = rgb2gray(enter);
enter2_g = rgb2gray(enter2);
leaving_g = rgb2gray(leaving);
leaving2_g = rgb2gray(leaving2);
enterandexit_g = rgb2gray(enterandexit);
enterandexit2_g = rgb2gray(enterandexit2);
exitan
terandenter_g = rgb2gray(exitan
terandenter2_g = rgb2gray(exitan

% Resize to scale (Normalize)
enter_g = imresize(enter_g, [xscale_new yscale_new]);
enter2_g = imresize(enter2_g, [xscale_new yscale_new]);
leaving_g = imresize(leaving_g, [xscale_new yscale_new]);
leaving2_g = imresize(leaving2_g, [xscale_new yscale_new]);
enterandexit_g = imresize(enterandexit_g, [xscale_new yscale_new]);
enterandexit2_g = imresize(enterandexit2_g, [xscale_new yscale_new]);
exitan
terandenter_g = imresize(exitan
terandenter2_g = imresize(exitan

% Obtain HOG Features
res_enter = HOG(enter_g);
res_enter2 = HOG(enter2_g);
res_leaving = HOG(leaving_g);
res_leaving2 = HOG(leaving2_g);
res_enterandexit = HOG(enterandexit_g);
res-enterandexit2 = HOG(enterandexit2_g);
res-exitan
terandenter_g = HOG(exitan
terandenter2_g = HOG(exitan
%% Test a Given Input Image

%in_ant2 = importdata('patterns/data/sideToSide/2to3StSV2/2_sts_2.txt');
%in_ant3 = importdata('patterns/data/sideToSide/2to3StSV2/3_sts_2.txt');
%in_ant2 = importdata('patterns/data/sideToSideAndBack/3to2to3StSV3/2bf_3.txt');
%in_ant3 = importdata('patterns/data/sideToSideAndBack/3to2to3StSV3/3bf_3.txt');

in_ant2 = importdata(filename1);
in_ant3 = importdata(filename2);

% Obtain variable to plot
in_ant2end = length(in_ant2)
in_ant3end = length(in_ant3)

rssi_2 = in_ant2(1:in_ant2end, 2); rssi_3 = in_ant3(1:in_ant3end, 2);
time_2 = in_ant2(1:in_ant2end, 3); time_3 = in_ant3(1:in_ant3end, 3);

% Plot Values Obtained From Antenna 2 (and extract them...)
figure(1);
plot(time_2, rssi_2, 'red', 'LineWidth', 2);
set(gcf, 'color', 'w');
axis off;
f = getframe(1);
img2 = frame2im(f);
img_gray2 = rgb2gray(img2);

% Plot Values Obtained From Antenna 3 (and extract them..)
figure(2);
plot(time_3, rssi_3, 'blue', 'LineWidth', 2);
set(gcf, 'color', 'w');
axis off;
f = getframe(2);
img3 = frame2im(f);
img_gray3 = rgb2gray(img3);

figure(3)
plot(time_2, rssi_2, 'red');
hold on
plot(time_3, rssi_3, 'blue');

% Evaluate Matrix (Antenna2)
matrix = img2;
matrix_bw = im2bw(matrix);
matrix_bw = imresize(matrix_bw, [xscale_new yscale_new]);

% Evaluate Matrix (Antenna3)
matrix3 = img3;
matrix_bw3 = im2bw(matrix3);
matrix_bw3 = imresize(matrix_bw3, [xscale_new yscale_new]);
%% Normalize 2/11
%http://www.mathworks.com/matlabcentral/newsreader/view_thread/316725
%matriax(row,column)

% Top
    top = 0;
    for a = 1:1:xscale_new
        for b = 1:1:yscale_new
            % Look from top left to right
            if(matrix_bw(a,b) == 0)
                top = a;
                break;
            else
                ;
            end
        end
        if(a == top)
            break;
        end
    end

% Left
    left = 0;
    for a = 1:1:yscale_new
        for b = 1:1:xscale_new
            % Look from top left downwards to the right
            if(matrix_bw(b,a) == 0)
                left=a;
                break;
            else
                ;
            end
        end
        if(left ~= 0)
            break;
        end
    end

% Right
    right = 0;
    for a = yscale_new:-1:1
        for b = xscale_new:-1:1
            % Look from bottom right to the top to the left
            if(matrix_bw(b,a) == 0)
                right=a;
                break;
            else
                ;
            end
        end
    end
if(right ~= 0)
    break;
end
end

% Bottom
bottom = 0;
for a = xscale_new:-1:1
    for b = yscale_new:-1:1
        % Look from bottom left to the right and up
        if(matrix_bw(a,b) == 0)
            bottom = a;
            break;
        else
        end
    end
    if(a == bottom)
        break;
    end
end

% Rescale again.
matrix = rgb2gray(matrix);
matrix = imresize(matrix, [xscale_new yscale_new]);
scaled_img = matrix(top:bottom,left:right);

%% Repeat for 3:
% Top
    top = 0;
    for a = 1:1:xscale_new
        for b = 1:1:yscale_new
            % Look from top left to right
            if(matrix_bw3(a,b) == 0)
                top = a;
                break;
            else
            end
        end
        if(a == top)
            break;
        end
    end

% Left
    left = 0;
    for a = 1:1:yscale_new
        for b = 1:1:xscale_new
            % Look from top left downwards to the right
            if(matrix_bw3(b,a) == 0)
left=a;
break;
else
    ;
end
end
if(left ~= 0)
    break;
end
end

% Right
right = 0;
for a = yscale_new:-1:1
    for b = xscale_new:-1:1
        % Look from bottom right to the top to the left
        if(matrix_bw3(b,a) == 0)
            right=a;
            break;
        else
            ;
        end
    end
    if(right ~= 0)
        break;
    end
end

% Bottom
bottom = 0;
for a = xscale_new:-1:1
    for b = yscale_new:-1:1
        % Look from bottom left to the right and up
        if(matrix_bw3(a,b) == 0)
            bottom=a;
            break;
        else
            ;
        end
    end
    if(a == bottom)
        break;
    end
end

% Rescale again.
matrix3 = rgb2gray(matrix3);
matrix3 = imresize(matrix3, [xscale_new yscale_new]);
scaled_img3 = matrix3(top:bottom,left:right);

% %%
% View output in images
output = HOG(scaled_img);
output3 = HOG(scaled_img3);

% figure(1);
% bar(res_sts2);
% title('StS_ANT2 HOG');
%
% figure(2);
% bar(res_stsandb2);
% title('StSaB_ANT2 HOG');
%
% figure(3);
% bar(res_sts3);
% title('StS_ANT3 HOG');
%
% figure(4);
% bar(res_stsandb3);
% title('StSaB_ANT3 HOG');
%
% figure(5);
% bar(output);
% title('HOG of input Antenna2');
%
% figure(6);
% bar(output3);
% title('HOG of input Antenna3');
%
% figure(7);
% imshow(scaled_img);
%
% figure(8);
% imshow(scaled_img3);
%
% Conclusion:

% is_enter = mean(abs(output - res_enter));
% is_enter2 = mean(abs(output - res_enter2));
% is_leave = mean(abs(output - res_leaving));
% is_leave2 = mean(abs(output - res_leaving2));
% is_eAe = mean(abs(output - res_enterandexit));
% is_eAe2 = mean(abs(output - res_enterandexit2));
% is_lAe = mean(abs(output - res_exitandenter));
% is_lAe2 = mean(abs(output - res_exitandenter2));
%
% Compare Antenna 2
out_enter = cov(output, res_enter);
out_enter2 = cov(output, res_enter2);
out_leave = cov(output, res_leaving);
out_leave2 = cov(output, res_leaving2);
out_eAe = cov(output, res_enterandexit);
out_eAe2 = cov(output, res_enterandexit2);
out_lAe = cov(output, res_exitandenter);
out_lAe2 = cov(output, res_exitandenter2);

% list = [is_enter is_enter2 is_leave is_leave2 is_eAe is_eAe2 is_lAe is_lAe2]
list = [out_enter(1,2) out_leave(1,2) out_eAe(1,2) out_lAe(1,2)]

% winner = min(list);
winner = max(list);

if winner == list(1)
a=1; %Enter
elseif winner == list(2)
a=2; %Leaving
elseif winner == list(3)
a=3; %Enter and Exit
elseif winner == list(4)
a=4; %Exit and Enter
else
    ERROR = 'ERROR'
a=0
end

% Compare Antenna 3
% is_enter = mean(abs(output3 - res_enter));
% is_enter2 = mean(abs(output3 - res_enter2));
% is_leave = mean(abs(output3 - res_leaving));
% is_leave2 = mean(abs(output3 - res_leaving2));
% is_eAe = mean(abs(output3 - res_enterandexit));
% is_eAe2 = mean(abs(output3 - res_enterandexit2));
% is_lAe = mean(abs(output3 - res_exitandenter));
% is_lAe2 = mean(abs(output3 - res_exitandenter2));

out_enter = cov(output3, res_enter);
out_enter2 = cov(output3, res_enter2);
out_leave = cov(output3, res_leaving);
out_leave2 = cov(output3, res_leaving2);
out_eAe = cov(output3, res_enterandexit);
out_eAe2 = cov(output3, res_enterandexit2);
out_lAe = cov(output3, res_exitandenter);
out_lAe2 = cov(output3, res_exitandenter2);

% list2 = [is_enter is_enter2 is_leave is_leave2 is_eAe is_eAe2 is_lAe is_lAe2]
list2 = [out_enter(1,2) out_leave(1,2) out_eAe(1,2) out_lAe(1,2)]

% winner = min(list2);
winner = max(list2);

if winner == list2(1)
    b=1; %Enter
elseif winner == list2(2)
    b=2; %Leaving
elseif winner == list2(3)
    b=3; %Enter and Exit
elseif winner == list2(4)
    b=4; %Exit and Enter
else
    ERROR = 'ERROR'
    b=0
end

% A: Antenna 2
% B: Antenna 3
% 1: Entering
% 2: Exiting
% 3: Enter and Exit
% 4: Exit and Enter
if(a==1 && b==2)
    v=1; % Feature 1 next song
elseif(a==2 && b==1)
    v=2; % Feature 2 last song
elseif(a==3 && b==4)
    v=3; % Feature... 'n' speed up
elseif(a==4 && b==3)
    v=4;
elseif(a==3 && b==2)
    v=1;
elseif(a==1 && b==4)
    v=1;
elseif(a==3 && b==1)
    v=2;
else
    v=0;
end
a
b
v
end

Hog Feature Extraction
%Image descriptor based on Histogram of Orientated Gradients for gray-level images. This code
%was developed for the work: O. Ludwig, D. Delgado, V. Goncalves, and U. Nunes, 'Trainable
%Classifier-Fusion Schemes: An Application To Pedestrian Detection,' In: 12th International IEEE
function H=HOG(Im)
nwin_x=3; %set here the number of HOG windows per bound box
nwin_y=3;
B=9; %set here the number of histogram bins
[L,C]=size(Im); % L num of lines ; C num of columns
H=zeros(nwin_x*nwin_y*B,1); % column vector with zeros
m=sqrt(L/2);
if C==1 % if num of columns==1
    Im=im_recover(Im,m,2*m); %verify the size of image, e.g. 25x50
    L=2*m;
    C=m;
end
Im=double(Im);
step_x=floor(C/(nwin_x+1));
step_y=floor(L/(nwin_y+1));
cont=0;
hx = [-1,0,1];
hy = -hx';
grad_xr = imfilter(double(Im),hx);
grad_yu = imfilter(double(Im),hy);
angles=atan2(grad_yu,grad_xr);
magnit=((grad_yu.^2)+(grad_xr.^2)).^0.5;
for n=0:nwin_y-1
    for m=0:nwin_x-1
        cont=cont+1;
    end
    angles2=angles(n*step_y+1:(n+2)*step_y,m*step_x+1:(m+2)*step_x);
magnit2=magnit(n*step_y+1:(n+2)*step_y,m*step_x+1:(m+2)*step_x);
v_angles=angles2(:);
v_magnit=magnit2(:);
K=max(size(v_angles));
    %assembling the histogram with 9 bins (range of 20 degrees per bin)
    bin=0;
    H2=zeros(B,1);
    for ang_lim=-pi+2*pi/B:2*pi/B:pi
        bin=bin+1;
        for k=1:K
            if v_angles(k)<=ang_lim
                v_angles(k)=100;
                H2(bin)=H2(bin)+v_magnit(k);
            end
        end
    end
    H2=H2/(norm(H2)+0.01);
\[ H((\text{cont} - 1) \cdot B + 1: \text{cont} \cdot B, 1) = H_2; \]

\]

\end

end

\]

\]

\]

Data Communication (Java to Matlab)

% CLIENT connect to a server and read a message
% Usage - message = client(host, port, number_of_retries)
function message = readdata(d_input_stream, bytes_available)

import java.net.Socket
import java.io.*

number_of_retries = 2; % set to -1 for infinite
retry = 0;
message = [];
while true

    retry = retry + 1;
    if ((number_of_retries > 0) && (retry > number_of_retries))
        fprintf(1, 'Too many retries\n');
        break;
    end

    try
        fprintf(1, 'Retry %d connecting to %s:%d\n', ...
            %retry, host, port);
        % read data from the socket - wait a short
        time first
        pause(0.5);
        message = zeros(1, bytes_available, 'uint8');
        for i = 1:bytes_available
            message(i) = d_input_stream.readByte;
        end
        message = char(message);
        break;
    catch
        % pause before retrying
        pause(.1);
    end
end
function varargout = Music_RFID(varargin)
% MUSIC_RFID MATLAB code for Music_RFID.fig
% MUSIC_RFID, by itself, creates a new MUSIC_RFID or raises the existing
% singleton*.
% H = MUSIC_RFID returns the handle to a new MUSIC_RFID or the handle to
% the existing singleton*.
% MUSIC_RFID('CALLBACK', hObject, eventData, handles, ...)
% calls the local
% function named CALLBACK in MUSIC_RFID.m with the given input arguments.
% MUSIC_RFID('Property', 'Value', ...) creates a new MUSIC_RFID or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before Music_RFID_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to Music_RFID_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Music_RFID

% Last Modified by GUIDE v2.5 30-Apr-2014 02:41:16

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @Music_RFID_OpeningFcn, ...
    'gui_OutputFcn', @Music_RFID_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
end
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Music_RFID is made visible.
function Music_RFID_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Music_RFID (see VARARGIN)

% Choose default command line output for Music_RFID
handles.output = hObject;
if (exist('ant2.txt', 'file')==2)
    delete('ant2.txt');
end
if (exist('ant3.txt', 'file')==2)
    delete('ant3.txt');
end

ha = axes('units','normalized','position',[0 0 1 1]);
uistack(ha,'bottom');
I=imread('background.png');
hi = imagesc(I);
set(ha,'handlevisibility','off','visible','off')
% Update handles structure
guida(hObject, handles);

% UIWAIT makes Music_RFID wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Music_RFID_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global s %sampled data from music file
global Fwav %sampled rate in Hz
global Wavp
global state
global file2
global file3
global v
state=0;
% Get default command line output from handles structure
varargout{1} = handles.output;
function PLAY_Callback(hObject, eventdata, handles)
    global s %sampled data from music file
    global Fwav %sampled rate in Hz
    global Wavp
global file
    Wavp = 0;

    file = uigetfile('*.*', 'Select the Wave File'); %prompts for a file
    [s,Fwav] = audioread(file); %takes the sampled data, sampling frequency, and bits per sample
    handles.p = audioplayer(s,(Fwav)); %creates audioplayer object named handles.p from loaded wav file
    play(handles.p) %plays modified wav file.
    guidata(hObject,handles);
end

function LOAD_FILE_Callback(hObject, eventdata, handles)
    file2 = uigetfile('*.*', 'Select the Wave File'); %prompts for a file
    [v,Fwav] = audioread(file2); %takes the sampled data, sampling frequency, and bits per sample
    handles.p = audioplayer(v,(Fwav)); %creates audioplayer object named handles.p from loaded wav file
    play(handles.p) %plays modified wav file.

    % plot
    plot(handles.axes3,s) %plots wav signal
    title(handles.axes3,'Music Signal');xlabel(handles.axes3,'time');
    %labels for plot
    if (exist('ant2.txt', 'file')==2)
        delete('ant2.txt');
end
if (exist('ant3.txt', 'file')==2)
delte('ant3.txt');
end
while true
    if (exist('ant2.txt', 'file')==2) && (exist('ant3.txt', 'file')==2)
        close all
        file2=('ant2.txt');
        file3=('ant3.txt');
        v = finaldemo(file2,file3)
        if (v == 3)
            pause(handles.p);
            handles.p = audioplayer(s,(2*Fwav));
            resume(handles.p);
        elseif (v==1)
            if (Wavp ==0)
                num = sscanf(file,'%i');
            end
            Wavp = 1;
            if (Wavp==1)
                [a,b] = strtok(file,'.');
                c = a(end:end)
                num = str2num(c);
            end
            num = num+1;
            if (num > 9)
                num = 1;
            end
            str = sprintf('%d.wav',num);
            file = strcat('C:\Users\Chris\Downloads\Matlab\music_from_RFID\ ',str);
            [s,Fwav]=audioread(file); %takes the
    sampled data, sampling frequency, and bits per sample
    handles.p = audioplayer(s,(Fwav)); %creates
    audioplayer object named handles.p from loaded wav file
    play(handles.p) %plays modified wav file.
    elseif (v==2)
        if (Wavp ==0)
            num = sscanf(file,'%i');
        end
        Wavp = 1;
        if (Wavp==1)
            [a,b] = strtok(file,'.');
            c = a(end:end)
            num = str2num(c);
        end
        num = num-1;
        if (num == 0)
            num = 9;
        end
        else
str = sprintf('%d.wav',num);
file = strcat('C:\Users\Chris\Downloads\Matlab\music_from_RFID\ ',str);
[s,Fwav]=audioread(file); %takes the
sampled data, sampling frequency, and bits per sample
handles.p = audioplayer(s,(Fwav)); %creates
 audioplayer object named handles.p from loaded wav file
play(handles.p) %plays modified wav file.
elseif (v==4)
    if (state ==1)
        resume(handles.p); %resumes song if pause was pressed already
        state = 0;
    else
        pause(handles.p);
        state = 1; %stops playing the .wav file
    end
else
    warndlg('Gesture not recognized. Please try again.')
end
delete('ant2.txt','ant3.txt')
else
    pause(3)
end
%
if (Wavp ==0)
    num = sscanf(file,'%i');
end
Wavp = 1;
if (Wavp==1)
    [a,b] = strtok(file,'.')
    c = a(end:end)
    num = str2num(c);
end
num = num+1;
if (num > 9)
    num = 1;
end
str = sprintf('%d.wav',num);
file = strcat('C:\Users\Chris\Downloads\Matlab\music_from_RFID\ ',str);
[s,Fwav]=audioread(file); %takes the
sampled data, sampling frequency, and bits per sample
handles.p = audioplayer(s,(Fwav)); %creates
 audioplayer object named handles.p from loaded wav file
play(handles.p) %plays modified wav file.
else
    pause(3)
end
%}
end
% file2=uigetfile('*.txt','Select the antenna 2 File'); %prompts for ant2 file
% file3=uigetfile('*.txt','Select the antenna 3 File'); %prompts for ant3 file

guidata(hObject,handles);

% --- Executes on button press in RUN.
function RUN_Callback(hObject, eventdata, handles)
% hObject    handle to RUN (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global s %sampled data from music file
global Fwav %sampled rate in Hz
global Wavp
if (Wavp==0)
    handles.p = audioplayer(s,(Fwav)); %creates audioplayer object named handles.p from loaded wav file
else
    handles.p = audioplayer(Wavp,(Fwav));
end
play(handles.p) %plays modified wav file.
guidata(hObject,handles); %updates handles

% --- Executes on button press in PAUSE.
function PAUSE_Callback(hObject, eventdata, handles)
% hObject    handle to PAUSE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global state
if (state ==1)
    resume(handles.p); %resumes song if pause was pressed already
    state = 0;
else
    pause(handles.p); %stops playing the .wav file
    state = 1;
end
guidata(hObject,handles); %updates handles

% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
% hObject    handle to STOP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
stop(handles.p); %stops playing the .wav file
guidata(hObject,handles); %updates handles.p

% --- Executes on button press in SPEED_UP.
function SPEED_UP_Callback(hObject, eventdata, handles)
% hObject    handle to SPEED_UP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global s    %sampled data from music file
global Fwav %sampled rate in Hz
global Wavp
pause(handles.p);
handles.p = audioplayer(s,(1.5*Fwav));
resume(handles.p);
guidata(hObject,handles); %updates handles.p

% --- Executes on button press in SKIP.
function SKIP_Callback(hObject, eventdata, handles)
% hObject    handle to SKIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global file
global s    %sampled data from music file
global Fwav %sampled rate in Hz
global Wavp
if (Wavp ==0)
    num = sscanf(file,'%i');
end
Wavp = 1;
if (Wavp==1)
    [a,b] = strtok(file,'.'),
    c = a(end:end)
    num = str2num(c)
end
num = num+1;
if (num > 9)
    num = 1;
end
str = sprintf('%d.wav',num);
file = strcat('C:\Users\Chris\Downloads\Matlab\music_from_RFID\',str);
[s,Fwav]=audioread(file); %takes the sampled
data, sampling frequency, and bits per sample
handles.p = audioplayer(s,(Fwav)); %creates audioplayer object
named handles.p from loaded wav file
play(handles.p) %plays modified wav file.
guidata(hObject,handles); %updates handles.p
% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes2

% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

Normalization functions (Crop, Rescale, black/white)

function mat = bw_img(x)

temp = imread(x);
mat = im2bw (temp);
mat = imresize(mat, [500 500]);
end

function mat = crop(input)

top=0; bottom=0; left=0; right=0; xscale = 500; yscale =500;
input = imresize(input, [xscale yscale]);

% Top
for a = 1:1:xscale
    for b = 1:1:yscale
        % Look from top left to right
        if(input(a,b) == 0)
            top = a;
            break;
        else
            ;
        end
    end
    if(a == top)
        break;
    end
end

% Left
for a = 1:1:yscale
    for b = 1:1:xscale
% Look from top left downwards to the right
if(input(b,a) == 0)
    left=a;
    break;
else
    ;
end
if(left ~= 0)
    break;
end

% Right
for a = yscale:-1:1
    for b = xscale:-1:1
        % Look from bottom right to the top to the left
        if(input(b,a) == 0)
            right=a;
            break;
        else
            ;
        end
        if(right ~= 0)
            break;
        end
    end
end

% Bottom
for a = xscale:-1:1
    for b = yscale:-1:1
        % Look from bottom left to the right and up
        if(input(a,b) == 0)
            bottom=a;
            break;
        else
            ;
        end
        if(a == bottom)
            break;
        end
    end
end

new_mat = input(top:bottom,left:right);
mat = imresize(new_mat, [xscale yscale]);
end

Algorithm Test GUI
function varargout = patterntest(varargin)
% PATTERNTEST MATLAB code for patterntest.fig
% PATTERNTEST, by itself, creates a new PATTERNTEST or raises the existing
% singleton*.
% H = PATTERNTEST returns the handle to a new PATTERNTEST or the handle to
% the existing singleton*.
% PATTERNTEST('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in PATTERNTEST.M with the given input arguments.
% PATTERNTEST('Property','Value',...) creates a new PATTERNTEST or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before patterntest_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to patterntest_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help patterntest

% Last Modified by GUIDE v2.5 24-Feb-2014 19:15:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @patterntest_OpeningFcn, ...
    'gui_OutputFcn', @patterntest_OutputFcn, ...
    'gui_LayoutFcn', [], ..., ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before patterntest is made visible.
function patterntest_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to patterntest (see VARARGIN)

% Choose default command line output for patterntest
handles.output = hObject;

% Update handles structure
guida(hObject, handles);

% UIWAIT makes patterntest wait for user response (see UIRESUME)
% uwait(handles.background);

% --- Outputs from this function are returned to the command line.
function varargout = patterntest_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in upload.
function upload_Callback(hObject, eventdata, handles)
% hObject    handle to upload (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    name = uigetfile({'*.jpg;*.tif;*.png;*.gif','All Image Files';...    '*. '*,'All Files'},'Select an Image');
    name = strcat('patterns/',name);
    set(handles.imgnametxt,'String',name);
    pic = bw_img(name);
    axes(handles.img1);
    imshow(pic,'Parent',handles.img1);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
    % hObject    handle to compare (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    op = get(handles.option, 'String');
    op = op{get(handles.option, 'Value')};

    new_lib = init_lib('patterns/lib/');
    lib_count = size(new_lib);
    lib_count = lib_count(2);
    lib = cell(1, lib_count);

    switch op
        case '--- Algorithms ---'
            % Select an algorithm
            set(handles.reg, 'String', '-N/A-');
        case '1) Pixel Matching'
            set(handles.reg, 'String', '-N/A-');

            xscale = 500; yscale=500; totalsize = xscale*yscale;

            for x = 1:1:lib_count
                lib{x} = bw_img(new_lib{x});
                % Initializes lib: BW 500x500 IMG
            end

            OUTPUT_MESSAGE1 = 'L-Shape is the match';
            OUTPUT_MESSAGE2 = 'C-Shape is the match';
            OUTPUT_MESSAGE3 = 'V-Shape is the match';

            input = bw_img(get(handles.imgnametxt,'String'));

            scaled_img = crop(input);

            % Compare
            counter = 1;

            while(counter<=lib_count)
                matched_data = 0;

                % Compare to values in library
                for a = 1:1:xscale
                    for b = 1:1:yscale
                        if(scaled_img(a,b) == lib{counter}(a,b))
matched_data = matched_data + 1;
else
    ;
end
end

total_matched_percentage = ((matched_data / (totalsize)) * 100);
if (counter == 1)
    c_per = total_matched_percentage;
elseif (counter == 2)
    l_per = total_matched_percentage;
elseif (counter == 3)
    v_per = total_matched_percentage;
end

counter = counter + 1;
end

final = max([c_per l_per v_per]);
set(handles.percent, 'String', final);
axes(handles.img2);

if (final == c_per)
    imshow(lib{1}, 'Parent', handles.img2);
    set(handles.imgnname, 'String', 'C-Shape');
    set(handles.text5, 'String', '');
    set(handles.reg, 'String', '');
elseif (final == l_per)
    imshow(lib{2}, 'Parent', handles.img2);
    set(handles.imgnname, 'String', 'L-Shape');
    set(handles.text5, 'String', '');
    set(handles.reg, 'String', '');
elseif (final == v_per)
    imshow(lib{3}, 'Parent', handles.img2);
    set(handles.imgnname, 'String', 'V-Shape');
    set(handles.text5, 'String', '');
    set(handles.reg, 'String', '');
else
end

case '2) Scaling'
    % Library (hardcoded: need to work on this later)
xscale = 500; yscale = 500; totalsize = xscale*yscale;
for x = 1:1:lib_count
    lib{x} = bw_img(new_lib{x});
    % Initializes lib: BW 500x500 IMG
end

OUTPUT_MESSAGE1 = 'L-Shape is the match';
OUTPUT_MESSAGE2 = 'C-Shape is the match';
OUTPUT_MESSAGE3 = 'V-Shape is the match';

input = imread(get(handles.imgnametxt, 'String'));
input = im2bw(input);
input = imresize(input, [xscale yscale]);
scaled_img = crop(input);

% Initialize
xs = zeros(1,500); xs2 = zeros(1, 500); xs3 = zeros(1,500); xs4 = zeros(1,500);
ys = zeros(1,500); ys2 = zeros(1, 500); ys3 = zeros(1,500); ys4 = zeros(1,500);
% l_x = xs; c_x = xs; v_x = xs;
% l_y = ys; c_y = ys; v_y = ys;
% lib_x = {l_x, c_x, v_x};
% lib_y = {l_y, c_y, v_y};

img=scaled_img;
for x = 1:1:xscale
    if (x==1)
        xs(x) = 500 - sum(img(:,x));
    else
        xs(x) = 500 - sum(img(:,x)) + xs(x-1);
    end
end
for y = 1:1:yscale
    if (y==1)
        ys(y) = 500 - sum(img(y,:));
    else
        ys(y) = 500 - sum(img(y,:)) + ys(y-1);
    end
end

% Obtain Library Values
img2 = lib{2};

% Scaling Practice
for x = 1:1:xscale
    if (x==1)
        xs2(x) = 500 - sum(img2(:,x));
    else
        xs2(x) = 500 - sum(img2(:,x)) + xs2(x-1);
    end
end
for y = 1:1:yscale
    if (y==1)
        ys2(y) = 500 - sum(img2(y,:));
    else
        ys2(y) = 500 - sum(img2(y,:)) + ys2(y-1);
    end
end

img2 = lib{1};

% Scaling Practice
for x = 1:1:xscale
    if (x==1)
        xs3(x) = 500 - sum(img2(:,x));
    else
        xs3(x) = 500 - sum(img2(:,x)) + xs3(x-1);
    end
end

for y = 1:1:yscale
    if (y==1)
        ys3(y) = 500 - sum(img2(y,:));
    else
        ys3(y) = 500 - sum(img2(y,:)) + ys3(y-1);
    end
end

img2 = lib{3};

% Scaling Practice
for x = 1:1:xscale
    if (x==1)
        xs4(x) = 500 - sum(img2(:,x));
    else
        xs4(x) = 500 - sum(img2(:,x)) + xs4(x-1);
    end
end

for y = 1:1:yscale
    if (y==1)
        ys4(y) = 500 - sum(img2(y,:));
    else
        ys4(y) = 500 - sum(img2(y,:)) + ys4(y-1);
    end
end

cool = 1:1:500;

% Plot X:
axes(handles.img1)
plot(cool, xs, 'o', cool, xs2, 'red', cool, xs3, 'blue', cool, xs4, 'black')

% Plot Y:
axes(handles.img2)
plot(cool, ys, 'x', cool, ys2, 'red', cool, ys3, 'blue', cool, ys4, 'black')

set(handles.imname, 'String', 'Red: L -- Blue: C -- Black: V');

reg = [regression(xs, xs2) regression(xs, xs3) regression(xs, xs4)];
final = max(reg);
if (final == reg(1))
    set(handles.percent, 'String', 'L-Shape');
    set(handles.reg, 'String', final);
    set(handles.text5, 'String', 'Regression:');
elseif (final == reg(2))
    set(handles.percent, 'String', 'C-Shape');
    set(handles.reg, 'String', final);
    set(handles.text5, 'String', 'Regression:');
elseif (final == reg(3))
    set(handles.percent, 'String', 'V-Shape');
    set(handles.reg, 'String', final);
    set(handles.text5, 'String', 'Regression:');
else
end
end

case '3) HOG'
for x = 1:1:lib_count
    lib{x} = bw_img(new_lib{x});
    % Initializes lib: BW 500x500 IMG
end

xscale_new = 640; yscale_new = 640;
temp = imread('patterns/lib/lshape2.png');
l_g = rgb2gray(temp);
temp = imread('patterns/lib/cshape2.png');
c_g = rgb2gray(temp);
temp = imread('patterns/lib/vshape2.png');
v_g = rgb2gray(temp);

l_g = imresize(l_g, [xscale_new yscale_new]);
c_g = imresize(c_g, [xscale_new yscale_new]);
v_g = imresize(v_g, [xscale_new yscale_new]);

resl = HOG(l_g);
resc = HOG(c_g);
resv = HOG(v_g);
input = imread(get(handles.imgnametxt,'String'));
input_bw = im2bw(input);
input_bw = imresize(input_bw, [xscale_new yscale_new]);

% Get parameters

% Top
top = 0;
for a = 1:1:xscale_new
    for b = 1:1:yscale_new
        % Look from top left to right
        if(input_bw(a,b) == 0)
            top = a;
            break;
        else
            ;
        end
    end
    if(a == top)
        break;
    end
end

% Left
left = 0;
for a = 1:1:yscale_new
    for b = 1:1:xscale_new
        % Look from top left downwards to the right
        if(input_bw(b,a) == 0)
            left=a;
            break;
        else
            ;
        end
    end
    if(left ~= 0)
        break;
    end
end

% Right
right = 0;
for a = yscale_new:-1:1
    for b = xscale_new:-1:1
        % Look from bottom right to the top to the left
        if(input_bw(b,a) == 0)
            right=a;
            break;
        else
            ;
        end
    end
if(right ~= 0)
    break;
end

% Bottom
bottom = 0;
for a = xscale_new:-1:1
    for b = yscale_new:-1:1
        % Look from bottom left to the right and up
        if(input_bw(a,b) == 0)
            bottom=a;
            break;
        else
        end
    end
    if(a == bottom)
        break;
    end
end

input = rgb2gray(input);
input = imresize(input, [xscale_new yscale_new]);
scaled_img = input(top:bottom, left:right);

output = HOG(scaled_img);

isc = output - resc;
isv = output - resv;
isl = output - resl;

% axes(handles.img1)
% subplot(2,2,1)
% bar(resc);
% title('HOG "C"');
% subplot(2,2,2)
% bar(resl);
% title('HOG "L"');
% subplot(2,2,3)
% bar(resv);
% title('HOG "V"');
% subplot(2,2,4)
% bar(output);
% title('HOG "Input"');
bar(isv);
title('Is it "V"');
subplot(3,1,3);
bar(isl);
title('Is it "L"');

% Obtain Covariance: Previously I used mean(___) 
out1 = cov(output, resc);
out2 = cov(output, resv);
out3 = cov(output, resl);

% out1 = abs(mean(isc));
% out2 = abs(mean(isv));
% out3 = abs(mean(isl));

list = [out1 out2 out3];
winner = max(list);
if (winner == list(1))
    set(handles.imgname, 'String', 'C-Shap
    axes(handles.img1)
    bar(resc);
    title('HOG Features of "C"');
    axes(handles.img2)
    bar(isc);
    title('Difference of Input and "C"');
    set(handles.text5, 'String', 'Var');
    set(handles.reg, 'String', winner);
    set(handles.percent, 'String', '');
elseif winner == list(2)
    set(handles.imgname, 'String', 'V-Shap
    axes(handles.img1)
    bar(resl);
    title('HOG Features of "V"');
    axes(handles.img2)
    bar(isl);
    title('Difference of Input and "V"');
    set(handles.text5, 'String', 'Var');
    set(handles.reg, 'String', winner);
    set(handles.percent, 'String', '');
elseif winner == list(3)
    set(handles.imgname, 'String', 'L-Shap
    axes(handles.img1)
    bar(resv);
    title('HOG Features of "L"');
    axes(handles.img2)
    bar(isv);
    title('Difference of Input and "L"');
    set(handles.text5, 'String', 'Var');
    set(handles.reg, 'String', winner);
    set(handles.percent, 'String', '');
else
end

otherwise
    % Error Message
    set(handles.reg, 'String', 'ERROR');
end

% --- Executes on selection change in option.
function option_Callback(hObject, eventdata, handles)
    % hObject    handle to option (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % Hints: contents = cellstr(get(hObject,'String')) returns option
    %        contents{get(hObject,'Value')} returns selected item from
    %        option

% --- Executes during object creation, after setting all properties.
function option_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to option (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns
    % called
    % Hint: popupmenu controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
                       get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
References


