

UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

A Pattern Recognition Framework for Embedded Systems

Permalink

<https://escholarship.org/uc/item/3s43z91m>

Author

Salehian, Shayan

Publication Date

2018

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

A Pattern Recognition Framework for Embedded Systems

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Shayan Salehian

March 2018

Thesis Committee:

Dr. Frank Vahid, Chairperson

Dr. Eamonn Keogh

Dr. Philip Brisk

Copyright by
Shayan Salehian
2018

The Thesis of Shayan Salehian is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to give my sincere appreciation to my advisor Prof. Frank Vahid for his constant support and assistance throughout my master's degree program. This thesis should not have been possible without Prof. Vahid's guidance and suggestions.

Moreover, I would like to thank Prof. Eamonn Keogh and Prof. Philip Brisk for serving on my thesis committee. I would also like to thank Alex Edgcomb for his help and valuable feedback in designing the experiment, Joe Allen and Bailey Herms for helping me through this study in different manners. I am also grateful to Prof. Jeffrey McDaniel for letting me conduct multiple rounds of experiments in his class and would like to express my gratitude to all of the participants.

Finally, I would like to thank my family and friends for their endless love and support throughout my life.

ABSTRACT OF THE THESIS

A Pattern Recognition Framework for Embedded Systems

by

Shayan Salehian

Master of Science, Graduate Program in Computer Science

University of California, Riverside, March 2018

Dr. Frank Vahid, Chairperson

Embedded systems are small computers dedicated to performing a specific task and can be designed as simple as a temperature controller to a complex medical imaging system. Embedded systems are ubiquitous having diverse applications in areas such as personal devices, factory automation, military, and medicine. A particular need in many embedded systems is recognizing patterns from available information to achieve a goal, such as determining the kind of fruit passing on a conveyor belt. Pattern recognition is a mature field that studies algorithms for learning patterns in data. However, many embedded systems designers do not have the expertise in the pattern recognition domain which imposes a challenge on employing these algorithms in their system designs. In this study, we introduce a pattern recognition framework for embedded systems that enables developers to use an interactive environment, tutorial, and reference code to develop K-nearest neighbors classification algorithm, which is a robust model in pattern recognition.

To validate benefits of the proposed framework, we conducted an experiment on 66 students to evaluate their performance in terms of the code quality and development speed when the framework is used, compared to when it is not. The results demonstrate a considerable gain in the development experience using our framework.

Table of Contents

List of Figures	viii
List of Abbreviations	x
1. Introduction	1
2. Related Work	4
2.1. Embedded Systems Applications	4
2.2. Classification	6
2.3. Embedded Systems and Classification	6
3. Classification Techniques for Embedded Systems Development	7
3.1. K-Nearest Neighbors	8
3.2. Logistic Regression	9
3.3. Naive Bayes	10
3.4. Decision Tree	10
3.5. Support Vector Machine	11
3.6. Neural Network	12
3.7. Summary	13
4. Framework	14
5. Experiment	27
5.1. Experimental Setup	27
5.1.1. Platform	27
5.1.2. Participants	34
5.3. Data Analysis and Results	34
5.3.1. Code Performance and Solving Speed	35
5.3.2. Experience Survey Results	37
6. Conclusion and Future Direction	44
7. References	45

List of Figures

Figure 1. Overlap of mature domains with embedded systems.	2
Figure 2. Pattern recognition based on sensor data collected from items on the conveyor belt.	3
Figure 3. Screenshot of original RIMS.	15
Figure 4. Screenshot of our framework which is an augmented version of RIMS.	16
Figure 5. Three main stages of pattern recognition in the reference code.	19
Figure 6. Screenshot of the graphical visualization of the fruits classification problem.	26
Figure 7. Screenshot of the first page of the experiment website.	29
Figure 8. Screenshot of the main page of the experiment website.	31
Figure 9. The training data in 2D dimensions.	32
Figure 10. Screenshot of the survey of the experiment.	33
Figure 11. The students participating in one of the three experiment sessions.	34
Figure 12. Accuracy of the designed models by group A, the group with access to pattern recognition resources.	36
Figure 13. Accuracy of the designed models by group B, the group with no access to pattern recognition resources.	37
Figure 14. Responses of group A and B to the first survey question.	38
Figure 15. Responses of group A and B to the second survey question.	39

Figure 16. Responses of group A and B to the third survey question.	40
Figure 17. Responses of group A and B to the fourth survey question.	41
Figure 18. Responses of group A and B to the fifth survey question.	42
Figure 19. Responses group A and B to the sixth survey question.	43
Figure 20. Responses of group A and B to the seventh survey question.	44

List of Abbreviations

PID	Proportional-integral-derivative
FIR	Finite impulse response
KNN	K-nearest neighbors
SVM	Support Vector Machine
DT	Decision Tree
LR	Logistic Regression
NN	Neural Network
UCR	University of California Riverside

1. Introduction

Embedded systems were initially introduced in the 1980s as task-oriented systems that had hardware and software modules embedded inside. The hardware module consisted of microprocessors, memories, and input/output units, and the software module was generally designed as an ad hoc assembly-level program to address the specific purpose of the system. Over time, embedded system applications became increasingly popular and diverse due to the reduction in the price and size of microcontrollers and the rise in microcontrollers' processing power and functionality. These changes led embedded systems to employ more advanced model-based designs and to evolve into systems with separated software and hardware modules and real-time capabilities. Following these advancements, embedded system development frameworks such as Arduino, Raspberry Pi, and Embedded Makers were introduced which also brought popularity to such systems.

Today, ninety-eight percent of all microprocessors are manufactured as components of embedded systems [1]. The most familiar examples of embedded systems are personal devices such as smartwatches and personal music players. However, the application scope is much wider and ranges from simple controllers such as those used in cooking and traffic lights to more advanced areas such as medicine, factory controllers, commerce, and military. The complexity of embedded systems varies from low, with a single microcontroller chip, to high with multiple units and tens of millions of lines of software.

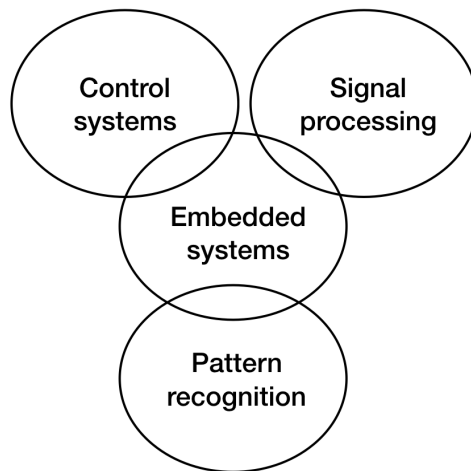


Figure 1. Overlap of mature domains with embedded systems.

In many embedded systems applications, there is an overlap with other mature domains such as control systems, signal processing, and pattern recognition (Figure 1). Therefore, in these overlapping applications, embedded systems developers may need to consult with experts in the other fields to create a product [2, 3]. However, many embedded systems developers do not have the budget or access to these experts, nor have the knowledge of the other domains. As a solution, for some domains that are widely adopted in embedded system development, such as control systems and signal processing, a reusable framework has been created by the domains' experts. These frameworks, such as proportional-integral-derivative (PID) controllers and finite impulse response (FIR) filters, facilitate the design of embedded systems for developers without the need for expert domain knowledge.

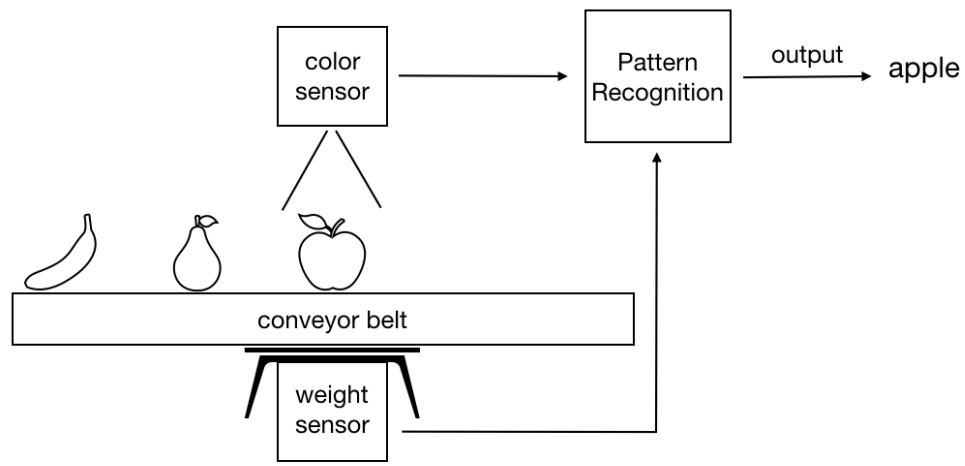


Figure 2. Pattern recognition based on sensor data collected from items on the conveyor belt.

In the past decade, pattern recognition has shown great advantages when adopted in different fields and can be incorporated into many embedded systems to improve the performance or introduce new applications [4, 5]. For instance, a pattern recognition module alongside with a color and weight sensor can be embedded into a conveyor belt to allow automatic detection of fruit types in a factory and automate this process (see Figure 2). Therefore, like the control systems and signal processing domains, there is a need to have a reusable discipline for the pattern recognition field in embedded systems. Such a discipline would pave the way for adopting and improving the application of pattern recognition in embedded systems.

In this project, we introduce a framework for pattern recognition to be used by general embedded systems developers. The contributions of this project are as follows:

- Identifying pros and cons of different pattern recognition techniques for embedded systems development and specifying the most suitable one for general embedded systems developers.
- Developing a convenient and adaptable framework to empower embedded systems developers to easily implement pattern recognition algorithms.
- Designing reference code and a tutorial to be used by developers.
- Conducting an experiment that evaluates the performance of developers in presence and absence of our framework.

2. Related Work

In this section, we first review embedded system applications through the past years.

Then, we provide a background on pattern recognition and classification algorithms and their success in recent years. Finally, we study the application of pattern recognition in embedded system design.

2.1. Embedded Systems Applications

Since the advent of digital technology that took place in the 1970s/1980s, the mass production of digital technologies and circuits has advanced and continued to this day. A central part of these advancements has been introduction of microcontrollers, which are small computers on a single integrated circuit. Microcontrollers allowed researchers and engineers to embed computing power into various devices and controllers, called embedded systems. Such systems are generally designed as low cost and performance-

centric devices to perform a specific task in real-time and, if needed, interact with the environment using a set of sensors. Nowadays, embedded systems are ubiquitous and their applications are diverse, ranging from personal electronic devices to more advanced devices used in industry, automation, aerospace, and medicine [6]. Daily used electronic devices with embedded computing power allow more flexibility in function and control features. For instance, washing machines, ovens, music players, and smart cameras all include embedded processors. More examples include personal devices that are common these days, such as smartwatches, activity trackers, and devices with augmented sensing such as a smartwatch with gesture and object recognition capabilities [7]. In industrial, automation, and aerospace domains embedded systems are used as controllers to bring partial or total automation. Examples include automatic gas detection and alert system [8], wall climbing robots that employ internal processors to control actions of the robot [9], and smart surveillance cameras [10]. On top of embedded control systems, processing of signals in an embedded processor is also common in diverse applications especially in medicine. A heart pulse monitoring system using air pressure and ultrasound signal processing [11], biomedical sensing such as blood testing devices [12], advanced DNA detection [13], and respiratory monitoring systems [14] are examples of health applications of embedded systems.

Our work does not aim to replace such systems, but rather to enable more developers to build such systems without extensive expertise. Furthermore, developers of such previous systems may benefit as well from our framework if it may lead to more modular, maintainable code.

2.2. Classification

Pattern recognition has gained attention in recent years due to superior performance in many applications compared to conventional methods. A major set of pattern recognition algorithms are dedicated to the classification problem, which tries to identify the class that an object belongs to, from a set of predefined classes. For example, classes may include cats, dogs, and humans, and a given object is then classified as one of those. The objects in such problems are commonly described by a set of features such as color or size, and the classifier is modeled based on a training set in which features and the true class of objects are provided. Therefore, the main idea behind a classification algorithm is to find a decision boundary on objects' feature sets that separates different classes as accurately as possible. Classification algorithms have shown great performance in various areas of research in recent years. Vision, speech recognition, handwriting recognition, natural language processing, and recommender systems are examples of areas that have grown significantly after the rise of pattern recognition and classification techniques.

2.3. Embedded Systems and Classification

Recent applications of embedded systems have adopted classification techniques to detect an event of interest, classify data, or predict future events. Examples include smart traffic prediction [15], flood prediction [16], and low-cost color detection systems [17].

Health-related predictions embedded inside personal devices have also gained attention in recent years. Shi et al [18] uses accelerometer and gyroscope sensors data to

predict a subject's fall event using classification techniques. Kong et al. [19] embeds air pressure sensors in the shoes of subjects and employs pattern recognition methods for gait monitoring purposes. To monitor activity levels, Lee et al. [20] proposed a smart shirt with embedded accelerometer and gyroscope sensors. Other activity recognition devices such as Fitbit and smartwatches also employ the same set of sensors alongside with classification models to predict activities of a subject [21]. The MyHeart project adopts smart clothes to fight cardiovascular diseases and provide early diagnosis [22].

3. Classification Techniques for Embedded Systems Development

Although various classification algorithms have been introduced, not all of them are suitable for incorporation in an embedded system development framework. Widely adopted algorithms include K-nearest neighbors (KNN) [23], Logistic Regression (LR) [24], Naive Bayes (NB) [25], Decision Tree (DT) [26], Support Vector Machine (SVM) [27], and Neural Network (NN) [28]. An approach should possess two main features to be desirable for embedded systems. First, it should be robust and reasonably competitive with the conventional algorithms used in embedded systems. Second, it should be easy to grasp by embedded system developers, to modify and adjust it to their needs. Third, it should fit into embedded systems constraints. Therefore, when evaluating different approaches, features such as robustness, implementation complexity, concept complexity, number of parameters to be set, performance in different areas, memory usage, and generalization capacity are points of consideration. In the rest of this section, we review

established classification techniques along with their primary features, and more importantly, their advantages and disadvantages when applied in embedded systems.

3.1. K-Nearest Neighbors

KNN is a simple and effective algorithm that classifies a new input point by finding K closest points from the training set to the input point and then choosing the majority class of those points as the class of the input point. Specifying parameter K and a distance metric (such as Euclidean or Manhattan distance functions) to measure the distance of two points is key in this algorithm.

Advantages:

- KNN is simple both in concept and implementation.
- KNN is robust in performance. It is theoretically provable that in case of having a large enough training set, the error rate of 1-nearest neighbor would be less than twice of the minimum achievable error rate called Bayes error rate.
- KNN is inherently capable of multi-class classification.
- KNN can handle non-linearity of the input data.

Disadvantages:

- KNN needs to store all the training set for classification purposes and search through the training set per execution. In case of having a large training set, which is essential for some complex applications, this makes the classification process slow [29] and may use much memory.
- In the presence of high dimensional feature vectors for data points, the curse of

dimensionality problem [30] would happen, especially when Euclidean distance metric is employed.

3.2. Logistic Regression

LR is a linear binary classification algorithm. It is represented by an equation similar to linear regression and statistically predicts the odds of an object being in a class based on a number of predictor features.

Advantages:

- LR is fast and easy to implement for production.
- LR is powerful when proper transformations on the input feature vectors are specified to achieve a linear relationship among predictive features.

Disadvantages:

- LR assumes no error in the input. Therefore, noises and outliers should be removed from the training data to achieve good performance. This is while real-world data is usually noisy and cleaning the data requires a great amount of effort from embedded system developers and is not possible in some cases.
- LR assumes a linear relationship among input features which may not always be present. Furthermore, data transformation techniques, if possible, are not straightforward for embedded system developers to force linearity among features.
- LR face the convergence problem when the data is sparse and highly correlated.

3.3. Naive Bayes

A Naive Bayes classifier is designed based on the Bayes' theorem and learns the distribution of the input data to predict the probability that an input object belongs to a particular class.

Advantages:

- NB is understandable and simple to implement.
- NB works well with a small training dataset if the data for all classes and features are balanced.
- NB is not sensitive to noise.

Disadvantages:

- NB makes a strong assumption of independence between features of the input data. This assumption in real-world applications is generally violated.
- NB performance considerably suffers when the data is sparse in a number of classes or features. In many of the real-world applications, input data is imbalanced in the outcome and contains sparsity in one or more of their features.
- NB is not capable of handling continuous features directly.

3.4. Decision Tree

DT is a non-linear classifier which consists of a number of nodes that are either decision nodes or class nodes. All the nodes in the tree are decision nodes, except for the leaf nodes that each represents a particular class. For an unknown input data, starting from the root, at each decision node a feature is employed as the decision maker along with a

feature value threshold which determines the direction of the branch to the next child node. After a series of comparisons, the algorithm reaches a leaf node which determines the predicted class.

Advantages:

- DT is capable of providing interpretation of the decision-making process.
- DT can handle non-linearity of data and correlation of features.

Disadvantages:

- DT is unstable and has a high variance which makes it un-robust for classifying future observed data.
- DT does not usually work well when boundaries of classes are smooth.
- An ensemble of DTs that employs many DTs to increase the robustness of predictive model is complex and heavy for embedded system development.

3.5. Support Vector Machine

SVM is a sophisticated, yet simple to implement, binary classification algorithm, which performs classification by finding the hyperplane that represents the largest separation between two classes and maximizes their margin. The vectors that are at the margin of each class and define the hyperplane are called support vectors. The appropriate kernel function and the soft margin parameter C , are two main parameters of SVM model that needs to be tuned based on the task.

Advantages:

- SVM model only needs to store support vectors to make an accurate prediction in the production time, which is beneficial when the training data is large.
- SVM allows modeling of linear and non-linear data by changing the kernel function.
- SVM is robust in prediction due to maximizing the prediction margin and has shown good performance in recent embedded system applications such as mobile human airbag system [31], Human Action Recognition [32], and Activity Recognition based on acceleration data [33].

Disadvantages:

- SVM performance heavily relies on the kernel function and C parameter which requires extensive experiments to determine. Kernel functions used in SVM have various constraints and are not easy to choose especially for a non-expert developer.
- SVM has long training time compared to other models.
- SVM is inherently a binary classification algorithm, and its expansion to multi-class classification using one-against-one or one-against-all approaches introduces a high increase in training time.

3.6. Neural Network

NNs are inspired by the human brain and nervous system and have achieved much of state of the art in the image processing and natural language processing fields.

Advantages:

- NN is accurate and can classify complex distribution of data.

-NN can model both linear and non-linear input data.

Disadvantages:

- NN requires a large amount of data to be trained and achieve good prediction performance.

- NN is slow to train and relatively slow in production.

3.7. Summary

Considering discussed advantages and disadvantages of each algorithm and recognizing our three goals of simplicity, high-performance, and fitting embedded systems constraints, we chose KNN as the best algorithm to be employed by general embedded systems developers. NNs are somewhat complicated to understand, define, and implement. Moreover, they require a large amount of data to train. SVM is not easy to employ by non-experts due to its reliance on kernel functions to model non-linearity of data and parameter tuning requirements. NB and LR classifiers seem not suitable due to lack of generality by making independence and linearity assumptions on the input data. Finally, DTs lack robustness when applied to real-world data. An ensemble of DTs is also complicated to be modified and adopted by embedded systems developers.

KNN provides simplicity, robustness, and high accuracy. Moreover, its disadvantages do not hinder its application in embedded systems. To tackle the problem of storing large datasets for KNN classification, in most cases a small or aggregated training dataset can be employed. Moreover, embedded systems do not commonly

include a wide set of features, so the curse of dimensionality problem is not an issue in many embedded systems applications.

4. Framework

Providing a convenient and rich environment for embedded systems developers to implement pattern recognition algorithms is one of the primary goals of this project. Aiding this goal would be a framework with straightforward and handy tools to implement, compile, run, debug, and test code as well as tutorials of pattern recognition techniques. Having this in mind, we developed an online framework, which is built on top of the Riverside-Irvine Microcontroller Simulator (RIMS) [34]. RIMS is a modern web-based easy-to-use graphical environment for writing, compiling, executing, and debugging C code for the RI (virtual) microcontroller. RIMS is implemented in Python and Flask web framework on the server side and is designed by HTML, CSS, and Javascript on the client side. Figure 3 shows the user interface of RIMS. As it can be observed, the user interface enables users to write code in the text editor, input a one-byte value using the 8-bit toggle-buttons, compile, run, and receive a one-byte output value as 8 bit LEDs. The compile process in RIMS starts with sending the code to the server using an HTTP POST request. A web server and Flask web framework then receive the code and convert it into MIPS instructions, sending it back to the client side. Afterward, the code is ready to run on the client side. An efficient well-developed Javascript program runs MIPS instructions line by line in the browser and displays outputs to the user. This

approach enables the user to change input values in real time and observe outputs, without further communication with the server or click the run button again.

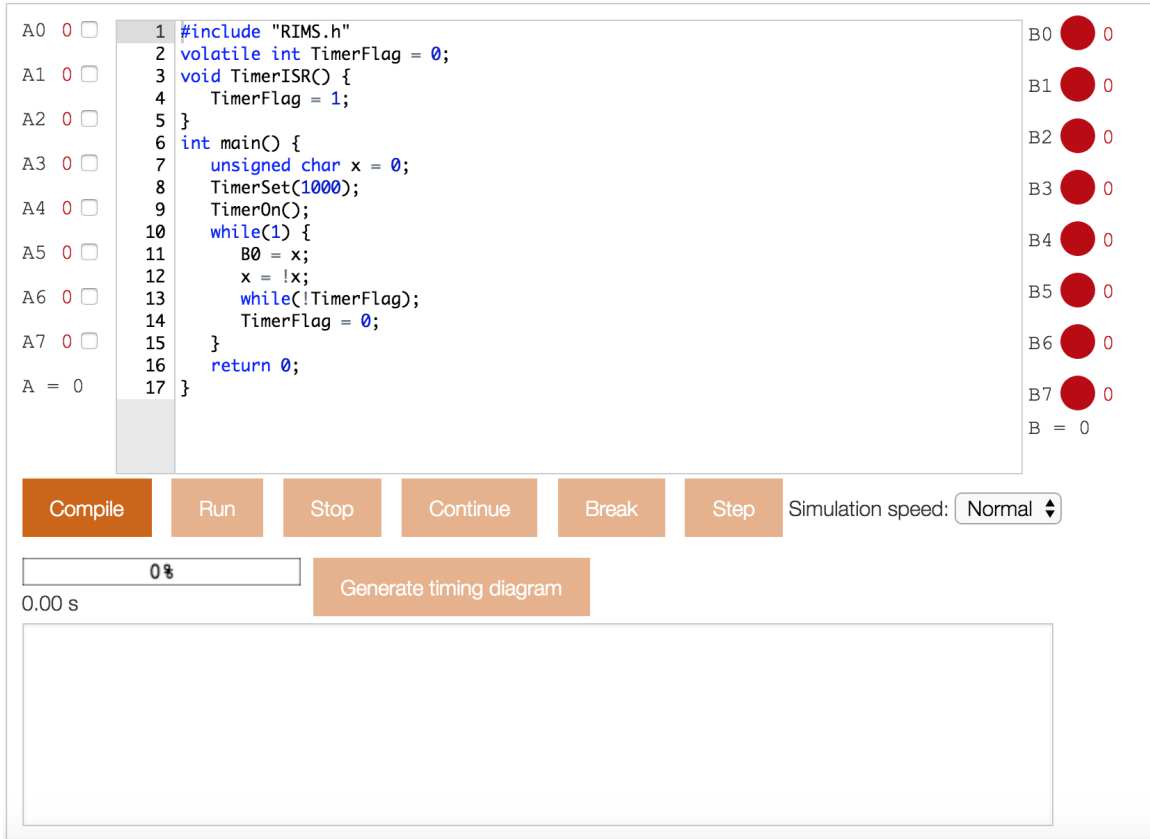


Figure 3. Screenshot of original RIMS.

While exploiting existing capabilities of RIMS, we improved it significantly to address the needs of the aimed framework. On the client side of the RIMS, we approximately added 3000 lines of code to the existing 8500 lines and modified around 450 lines of code to implement our improvements. In addition, on the server side of RIMS, we added around 200 and modified 20 lines of code from the existing 950 lines of code. Figure 4 shows the new design of RIMS which we employ in our proposed

framework. In the remaining part of this section, we review the main contributions that we have made to RIMS. These contributions not only enable us to build a pattern recognition framework for embedded system developers but also can be employed in other developments made on top of RIMS.

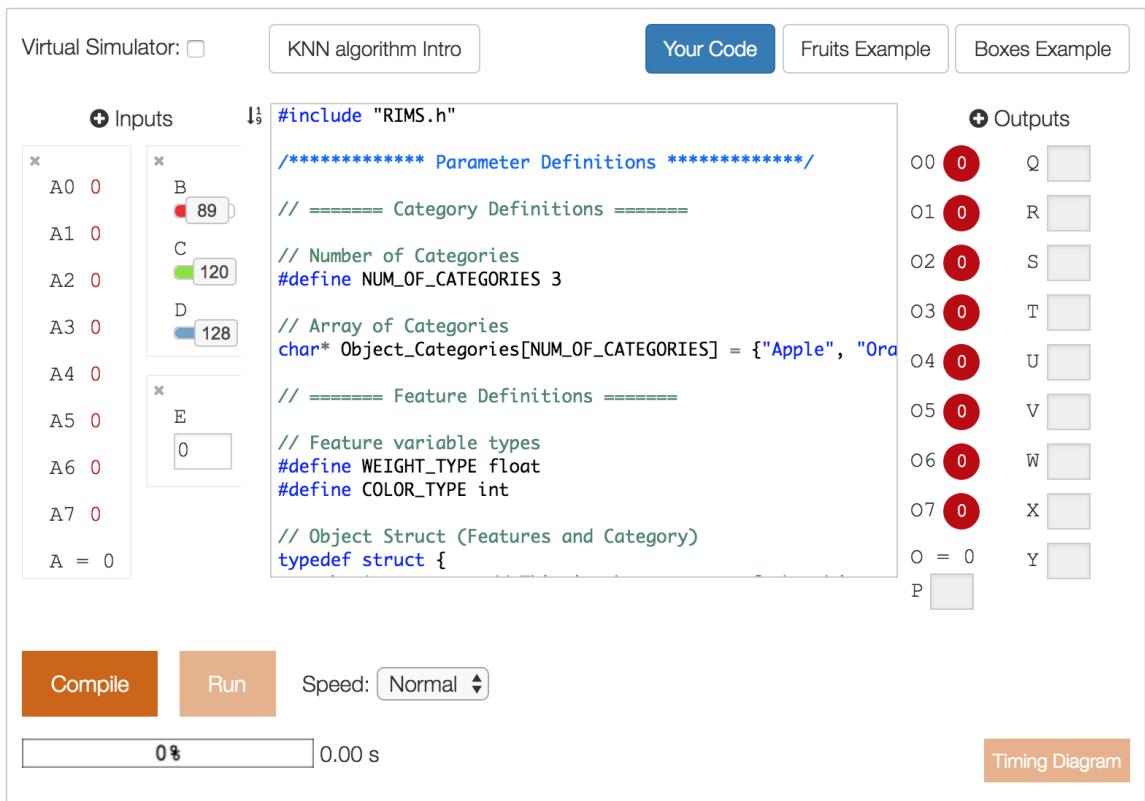


Figure 4. Screenshot of our framework which is an augmented version of RIMS.

1. Support for dynamic number of inputs

RIMS originally had a fixed one-byte input and output, bits of which were respectively named as $A_0, A_1, A_2, \dots, A_7$ and $B_0, B_1, B_2, \dots, B_7$. However, more

inputs and outputs are desirable in many embedded systems applications. Therefore, we added the ability to add or remove one-byte inputs and outputs dynamically. We reserved uppercase characters from A to N as variable names for inputs and from O to Z as variable names for outputs. Recognizing that each input and output is a one-byte value, each variable like A is represented by 8 bits: $A0$ to $A7$.

2. Support for various input and output types

Embedded systems developers deal with different types of input sensors and output actuators. To improve simulation of inputs and outputs, we provide different predefined input and output types based on the needs of developers when using widely adopted sensors. Input types include a bit-wise input for sensors such as break beam sensors, decimal input for any sensor that measures a decimal value, and color input for color sensors. Output types include a bit-wise output denoted by simulated LEDs and decimal output. In Figure 4, we can observe a bit-wise, a decimal, and a color input. Moreover, on the right-hand-side, we can see a bit-wise output alongside with nine decimal output actuators.

3. Providing reference code for easy classification development

To increase the development speed in our framework, we provide the full implementation of KNN algorithm for some sample problems and make those programs available to embedded systems developers as reference code so that they can solve their own problems just by modifying the code instead of writing code from scratch. We wrote the reference code in C language, which is a language used in

many embedded systems applications. Moreover, we put comments on the code to enable developers to easily navigate through the code and understand associated concepts and functions and find the desired sections for modification.

Our reference code divides pattern recognition into three main stages: feature extraction, classification, and actuation (see Figure 5). These stages are implemented in *FeatureExtraction*, *ClassifyKNN*, and *Actuation* functions respectively. Having each stage in a distinct code block allows easy modification and management for developers. In the feature extraction section, an object is created based on the current value of input sensors. In the classification section, KNN algorithm is implemented with a modifiable distance function and adjustable parameters. In the actuation section, an actuation function is implemented to show the classifier output as one of the output options provided by our enhanced version of RIMS. In addition to these sections, our reference code contains functions to add training data and normalize data.

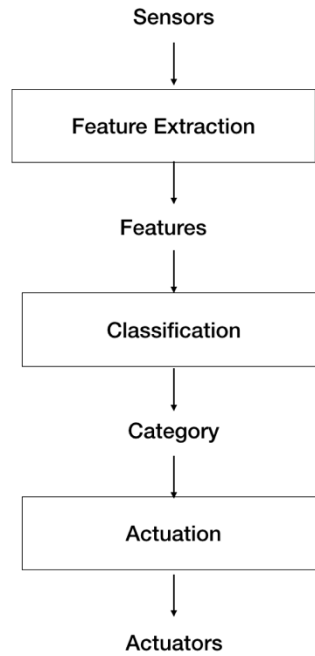


Figure 5. Three main stages of pattern recognition in the reference code.

We provide the reference code to developers as two classification examples: fruit and box classifications. The fruit classifier objective is to distinguish apples, oranges, and mandarins from each other using three sensors: color, weight, and break beam. Moreover, the goal of the box classifier is to classify boxes into small, medium, and large categories by utilizing a number of break beams. These sample classifications include all types of input data to provide a comprehensive reference to developers. The fruit classification example code is included below.

```

#include "RIMS.h"
#include <math.h>

/***** Parameter Definitions *****/

// ===== Category Definitions =====

// Number of Categories
#define NUM_OF_CATEGORIES 3

// Array of Categories
char* ObjectCategories[NUM_OF_CATEGORIES] = {"Apple", "Orange",
"Mandarin"};

// ===== Feature Definitions =====

// Object Struct (Features and the category)
typedef struct {
    char* category; // This is the category of the object
    float weight; // a feature
    float r; // a feature
    float g; // a feature
    float b; // a feature
} Object;

// Maximum and minimum values of features
const float WEIGHT_MAX = 150;
const float WEIGHT_MIN = 0;
const float COLOR_MAX = 255;
const float COLOR_MIN = 0;

// ===== Classification Definitions =====

// K parameter used in K-Nearest-Neighbors (KNN) Algorithm
const int K = 3;

// Number of Known Objects
#define NUM_OF_KNOWN_OBJECTS 9

// Array of Known Objects
Object knownObjects[NUM_OF_KNOWN_OBJECTS];

/***** Known Objects Preparation *****/

```

```

// ===== Normalization Util Functions =====

// Rescale a value to 0-1 range
float RescaleValue(float value, const float min, const float max) {
    return (value-min)/(max-min);
}

// Rescale the object features to 0-1 range
Object RescaleObject(Object object) {
    Object rescaledObject;
    rescaledObject.category = object.category;
    rescaledObject.weight = RescaleValue(object.weight, WEIGHT_MIN,
WEIGHT_MAX);
    rescaledObject.r = RescaleValue(object.r, COLOR_MIN, COLOR_MAX);
    rescaledObject.g = RescaleValue(object.g, COLOR_MIN, COLOR_MAX);
    rescaledObject.b = RescaleValue(object.b, COLOR_MIN, COLOR_MAX);
    return rescaledObject;
}

// ===== Populating known objects =====

// Add an object to the known objects array
void AddToKnownObjects(int i, char* category, float weight, float r, float
g, float b) {
    knownObjects[i].category = category;
    knownObjects[i].weight = weight;
    knownObjects[i].r = r;
    knownObjects[i].g = g;
    knownObjects[i].b = b;
    knownObjects[i] = RescaleObject(knownObjects[i]);
}

// Insert all Known objects into the array
void PopulateKnownObjects() {
    AddToKnownObjects(0, "Apple", 74.0, 159, 14, 13);
    AddToKnownObjects(1, "Apple", 87.0, 236, 57, 2);
    AddToKnownObjects(2, "Apple", 95.0, 175, 10, 34);

    AddToKnownObjects(3, "Orange", 135.0, 248, 118, 3);
    AddToKnownObjects(4, "Orange", 122.0, 241, 131, 21);
    AddToKnownObjects(5, "Orange", 131.0, 238, 128, 16);

    AddToKnownObjects(6, "Mandarin", 80.0, 244, 118, 11);
    AddToKnownObjects(7, "Mandarin", 75.0, 204, 90, 0);
}

```



```

    AddToKnownObjects(8, "Mandarin", 84.0, 228, 93, 28);
}

/***** Feature Extraction *****/

// Extract features from sensors (RIMS inputs) and create a new object
Object FeatureExtraction() {
    Object inputObject;
    inputObject.weight = A; // input "A" is weight
    inputObject.r = B; // input "B" is r
    inputObject.g = C; // input "C" is g
    inputObject.b = D; // input "D" is b
    return RescaleObject(inputObject);
}

/***** Classification (KNN) *****/

// ===== Util Functions =====

// Computes the euclidean distance between two objects.
float ComputeDistanceofObjects(Object object1, Object object2) {
    float weight = (object1.weight - object2.weight);
    float r = (object1.r - object2.r);
    float g = (object1.g - object2.g);
    float b = (object1.b - object2.b);
    float dist = sqrt(weight*weight + r*r + g*g + b*b);

    return dist;
}

// Sorts all the provided distances from small to large
void Sort(float *distances, char** categories) {
    int i;
    for (i = NUM_OF_KNOWN_OBJECTS - 1; i >= 0; --i) {
        int j;
        for (j = 0; j < i; ++j) {
            if (distances[i] < distances[j]) {
                float temp_dist;
                char* temp_category;
                temp_dist = distances[i];
                distances[i] = distances[j];
                distances[j] = temp_dist;
                temp_category = categories[i];
                categories[i] = categories[j];
                categories[j] = temp_category;
            }
        }
    }
}

```

```

    }
}
}

// ===== K-Nearest Neighbors (KNN) =====

// Implementation of KNN algorithm
// It takes an input object and a list of known objects and predicts the
category of the input object
char* ClassifyKNN(Object inputObject, Object knownObjects[]) {
    int count = 0;
    int max_count = 0;
    char* most_frequent_category;

    // Maintains K nearest knownObjects
    Object kNearestObjects[K];
    float distances[NUM_OF_KNOWN_OBJECTS];
    char* categories[NUM_OF_KNOWN_OBJECTS];

    int i;
    // Compute the distance of each known object to the input object
    for(i = 0; i < NUM_OF_KNOWN_OBJECTS; ++i) {
        int j;
        distances[i] = ComputeDistanceofObjects(inputObject,
knownObjects[i]);
        categories[i] = knownObjects[i].category;
    }

    // Sort distances in ascending order
    Sort(distances, categories);

    // Find out which category occurs most frequently among the K closest
known objects
    for(i = 0; i < NUM_OF_CATEGORIES; ++i) {
        int j;
        count = 0;
        // Check K closest ones
        for(j = 0; j < K; ++j) {
            if(categories[j] == ObjectCategories[i]) {
                count++;
            }
        }
        if(count > max_count) {
            max_count = count;
        }
    }
}

```

```

        most_frequent_category = ObjectCategories[i];
    }
}

return most_frequent_category;
}

/***** Output Preparation *****/

// Turn on the corresponding output bit to determine the category of the
input object
void Actuation(char* category) {
    if(category != "") {
        int i;
        for (i = 0; i < NUM_OF_CATEGORIES; ++i) {
            if( category == ObjectCategories[i] ){
                printf("%s\n", category);
                O = 0x01 << i;
            }
        }
    }
}

/***** Main Program *****/

volatile int TimerFlag = 0;

void TimerISR() {
    TimerFlag = 1;
}

int main() {
    PopulateKnownObjects();

    TimerSet(1000);
    TimerOn();

    while(1) {
        char *closest_object_category;

        // Feature Extraction
        Object inputObject = FeatureExtraction();

        // Classification
        closest_object_category = ClassifyKNN(inputObject, knownObjects);
    }
}

```

```
        // Actuation
        Actuation(closest_object_category);

        while(!TimerFlag);
        TimerFlag = 0;
    }

    return 0;
}
```

4. **Classification algorithm tutorial**


We provide a tutorial on the classification algorithm. Developers would understand the central concept of the algorithm both visually and by explanation. As a result, they would be more comfortable in working with the provided reference code.

5. **Graphical visualization of the reference code**

The reference code for box and fruit classification problems are accompanied by a graphical environment that enables developers to visually understand the problem and easily modify inputs to observe the impact on the output of the classification. Figure 6 shows the graphical visualization of the fruit classification example, which illustrates a table, a weight sensor, a color sensor, a number of break beams, as well as several sample fruits. Fruits can be dragged and dropped to the table to update values of the weight, color, and break beam sensors. In addition, the graphical interface provides the capability of importing and exporting sample objects in JSON format.

Virtual Simulator: KNN algorithm Intro Your Code Fruits Example Boxes Example

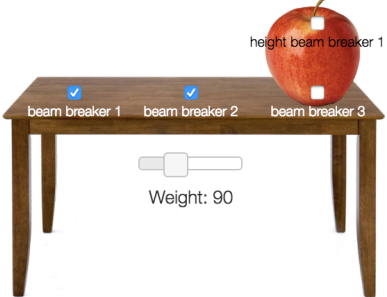
Import/Export



R: 179

G: 7

B: 35



Inputs

A0 0

A1 0

A2 0

A3 0

A4 0

A5 0

A6 0

A7 0

A = 0

```
#include "RIMS.h"

/***** Parameter Definitions *****/

// ===== Category Definitions =====

// Number of Categories
#define NUM_OF_CATEGORIES 3

// Array of Categories
char* ObjectCategories[NUM_OF_CATEGORIES] = {"Apple", "Oran

// ===== Feature Definitions =====

// Object Struct (Features and the category)
typedef struct {
    char* category; // This is the category of the object
    float weight; // a feature
    float r; // a feature
    float g; // a feature
```

Outputs

O0 0

O1 0

O2 0

O3 0

O4 0

O5 0

O6 0

O7 0

O = 0

Compile
Run
Speed: Normal

0%
0.00 s
Timing Diagram

Figure 6. Screenshot of the graphical visualization of the fruits classification problem.

5. Experiment

We conducted an experiment to evaluate the usability of our proposed framework and whether it leads to the development of higher quality programs or not. For this aim, we asked two groups of participants with similar knowledge background to implement a classification model in embedded systems. We provided one group with reference classification code and the classification tutorial, while we asked the other group to rely on their own knowledge to solve the problem. Then, we evaluated the performance of developers in terms of code correctness and speed.

5.1. Experimental Setup

5.1.1. Platform

To conduct the experiment, we designed an online website to guide participants, manage the process, and record the necessary information in a database. We used Node.js and Express web application framework for server-side developments and ReactJS, Jsx, and Sass for front-end developments. Also, we chose Postgresql as a general purpose and object-relational database management system to store data. We deployed the experiment website on Google Cloud Platform consisting of one Compute Engine instance and one SQL Storage instance.

Conducting the experiment through an online website has significant advantages compared to other forms of design such as a desktop application or simply making the framework accessible online. This approach allows us to minimize the verbal explanation

time, provide an equivalent environment for all the participants, and prevent possible misunderstandings. Furthermore, it enabled us to accurately record essential information from the beginning of the experiment to its end. The automatic distribution of participants into different groups was another notable advantage. It also let users participate in the experiment using their preferred device by just entering the IP address of our Google Cloud Compute Engine in their browser of choice.

The experiment website consists of three pages. Figure 7 shows the first page which is an introduction to the experiment containing greetings and important points about the goal and rules of the experiment. At this step, we have already assigned students to one of the two groups of A and B. Group A are provided with the pattern recognition resources while group B students do not have access to this information.



Welcome to RIMS Experiment

1 Question - 45 Mintues

- Welcome! We are so happy you are here!
- This experiment is designed to help us build a platform for improving the usability of embedded system development.
- Today, you will be asked to complete a coding experiment followed by a survey.
- **Your experiment might be different from your friends.** We are analyzing the impact of these differences on the final performance.
- Please **do not** use the internet to search, **do not** talk to your friends, and even **do not** look at each others' screens.
- All analysis will be done anonymously and your email only recorded to distinguish you from others.
- We want you to know that your participation has a big impact on advancement of this project and we are sincerely thankful in advance.

We'll never share your information with anyone else.

Start Experiment

Figure 7. Screenshot of the first page of the experiment website.

Figure 8 shows the main page of the experiment website which contains the problem statement and 18 training data samples. The control bars at the top of the page allow users to submit their code, view the experiment timer, and use environment tutorials based on their demand. The proposed framework is placed at the bottom of the page enabling participants to write, compile, and run their code online. The group A

participants have access to the KNN tutorial button in the framework while such access is not available for the group B.

We ask both groups to implement a solution that can make highly accurate predictions for a new data sample and evaluate each submitted code against one hundred test cases containing 18 provided training data and 82 new samples. We selected the problem (shown in Figure 8) such that a non-classification solution would not achieve high accuracy results and participants can easily figure this out by observing the overlap of data points inside different classes. Figure 9 illustrates the distribution of the training data provided to participants. To further analyze the experiment, we store a set of information for each participant including their email address, assigned group, final submitted code, and submission time.

Imagine you work as a tech developer in a big mall and you want to statistically analyze the number of men, women, and children entering the mall. To achieve this, a laser height sensor is placed on the ceiling and a weight sensor is placed on the floor at the entrance. These sensors capture the height of the customers in inches and the weight of the customers in pounds respectively. Moreover, an expert has manually recorded sensor readings for 18 people. You are now asked to develop a program to automatize this task.

We have provided a simulator for you to develop your program. You should add height and weight sensors as decimal inputs, named A and B respectively, and use them in your program by reading A and B as variables. Also, use LEDs (bit outputs) for showing the output by writing into the corresponding variables (O0: man, O1: woman, O2: child).

The 18 samples are provided for you below (The "categories" of the samples are labeled by an expert). Your code will be tested against these 18 samples and 82 other test cases not provided here. Try to write code as close as possible to what is being asked (it does not need to be completely accurate)

Samples (Category, Height, Weight):

"Woman", 58, 90	"Man", 69, 160	"Child", 41, 36
"Child", 45, 45	"Man", 71, 180	"Man", 68, 141
"Child", 55, 70	"Child", 48, 50	"Man", 75, 191
"Woman", 62, 110	"Woman", 70, 150	"Woman", 64, 99
"Woman", 64, 120	"Child", 58, 88	"Man", 63, 125
"Man", 62, 110	"Child", 60, 97	"Woman", 65, 121

Input/Output Example:

Input: **A (height) = 64, B (weight) = 125** --> Output: **O0 (man): 0, O1 (woman): 1, O2 (child): 0**
 Input: **A (height) = 75, B (weight) = 180** --> Output: **O0 (man): 1, O1 (woman): 0, O2 (child): 0**

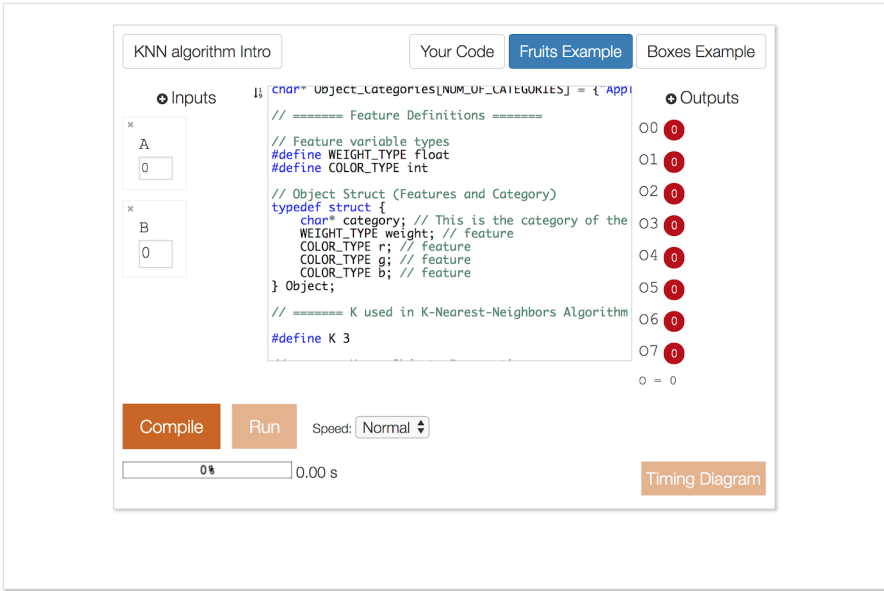


Figure 8. Screenshot of the main page of the experiment website.



Figure 9. The training data in 2D dimensions.

After receiving the participants’ developed programs, we guide them to an experience survey to evaluate our framework from different aspects, such as implementation usability, tutorials, and experiment setup. Figure 10 shows a snapshot of the survey which we developed by Google Forms. As the figure shows, we ask participants to select their level of agreement with seven statements. The first five statements are designed to evaluate participants’ experience in solving the problem and the last two ones are designed to collect their feedback on the development framework that we provided to them.

Regarding Solving the Problem *

	Strongly agree	Agree	Slightly agree	Slightly disagree	Disagree	Strongly disagree
I easily figured out how to solve the problem	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I easily implemented my solution	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I rarely thought of giving up	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The given time was enough to implement my solution	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I learned new methods to solve the problem	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Regarding the RIMS Tool *

	Strongly agree	Agree	Slightly agree	Slightly disagree	Disagree	Strongly disagree
I easily learned how to use the environment.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I easily used the environment to implement, compile, run, and test my program.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 10. Screenshot of the survey of the experiment.

5.1.2. Participants

All 66 students who were registered in CS122A class (Intermediate Embedded and Real-Time Systems) at UCR under the supervision of Prof. Jeffrey McDaniel in Fall 2017, participated in the study. We conducted the experiment across three lab sessions within two days. Because the experiment was designed as an online platform, students were allowed to use their laptops or lab computers to participate. Figure 11 shows students in the second group while participating in the experiment.



Figure 11. The students participating in one of the three experiment sessions.

5.3. Data Analysis and Results

In this section, we evaluate and compare the two groups of participants in terms of code quality and the problem-solving speed based on the information collected during the

experiment. After that, we analyze the feedback results collected from the experience survey.

5.3.1. Code Performance and Solving Speed

To evaluate the performance of models developed by participants, we employ accuracy evaluation metric which is defined as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ total\ predictions} \text{ (Eq. 1)}$$

The results show that the access to the pattern recognition resources in group A improved the accuracy of their models significantly (p -value=0.03). Specifically, group A achieved 71% accuracy on average while this value was 57% for group B. We can observe that the maximum achieved accuracy among all the models is 92%, and no one obtained 100% accuracy. This is because there is an overlap between different classes in the data. The accuracy distribution of group A, which is depicted in Figure 12, shows a bulge in the highest accuracy range, while the accuracy distribution of group B, illustrated in Figure 13, shows a more even distribution across all accuracy ranges. We can also observe that a number of group B models achieved less than 20 percent accuracy, while none of the models developed by group A were put in this range. This observation also validates the benefit of the proposed framework in guiding developers through solving classification problems in embedded systems. Also, a more detailed study of submitted models shows that all participants in group B used conditional statements to solve the problem. This is while all participants in group A decided to

modify the reference code to solve the problem although the default code provided to them was identical to group B.

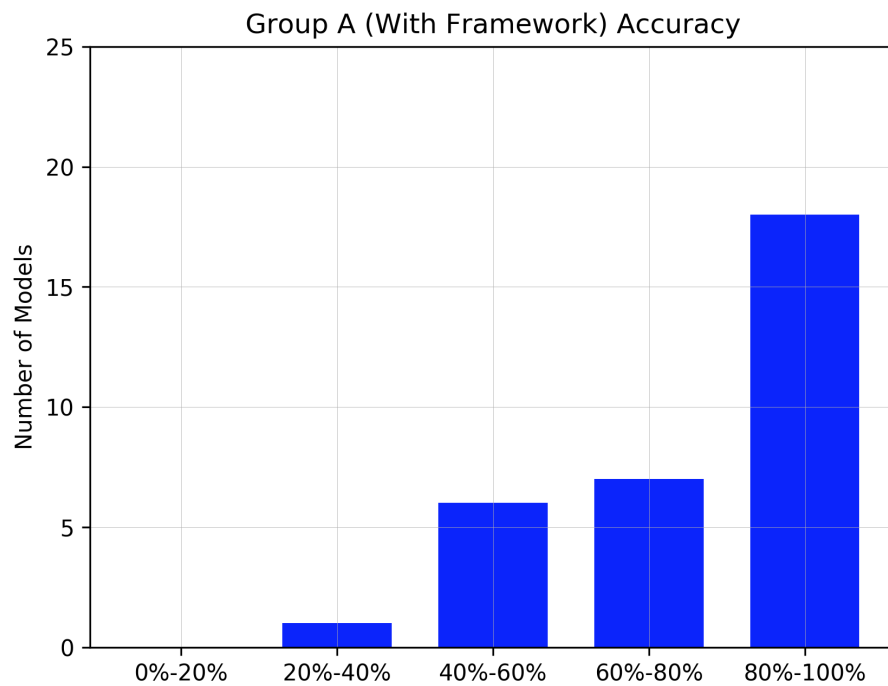


Figure 12. Accuracy of the designed models by group A, the group with access to pattern recognition resources.

Regarding the problem-solving speed, we analyze the submission time of participants in both groups. The participants were given 45 minutes to solve the problem. However, submission time information shows that group A spends 41 minutes on average to submit the solution which is slightly higher than the average submission time for group B which is measured 36 minutes. Although group A spends more time for solving the problem, they achieve much higher quality solutions. Moreover, group A needs to take time to read the tutorial and then review and understand the reference code in order to

modify it appropriately to solve the problem. The time spent on tutorials is a one-time overhead which will be reduced in future uses of the framework.

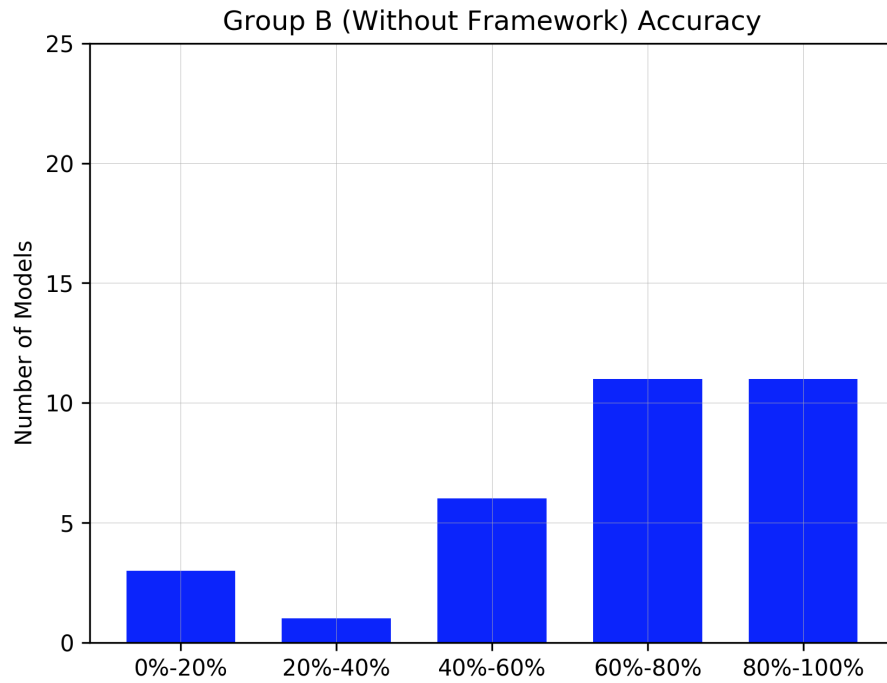


Figure 13. Accuracy of the designed models by group B, the group with no access to pattern recognition resources.

5.3.2. Experience Survey Results

Results of the experience survey provide us with important information regarding the usability of the proposed framework. Figures 14 to 20 show the statistics of the participant responses to the survey questions. The title of each figure contains the statement and the distribution of responses to each statement is displayed as blue and red bars for groups A and B respectively.

Figure 14 illustrates the distribution of responses to the first question, which reflects how easy each group has found solving the given problem. As it can be inferred, participants in group A feel more ease in approaching the problem. Moreover, our thorough study of submitted models shows that final solutions from group B are somewhat naive compared to group A. Therefore, we can conclude that the proposed framework results in more confident approach for developers especially when dealing with challenging classification problems and reduces the struggle in finding the suitable solution.

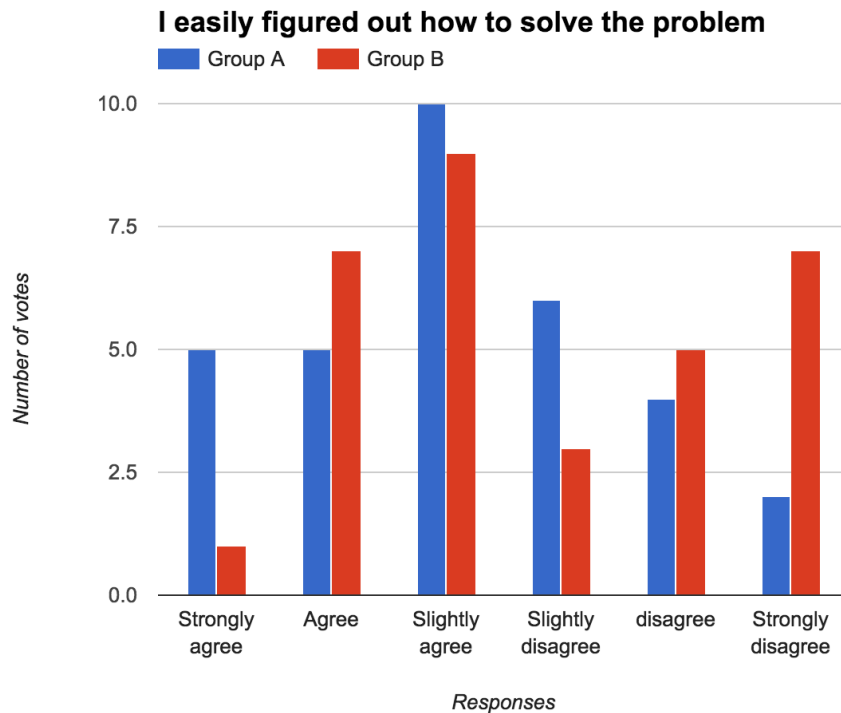


Figure 14. Responses of group A and B to the first survey question.

The second question aims to evaluate convenience of implementation for each group. Figure 15 demonstrates that the overall number of votes in the agree and disagree categories in both groups are very close. This observation confirms that developers in group A do not face extra hardship in the development of more complex models when compared to group B. Therefore, we can conclude that the pattern recognition resources provided in our proposed framework are beneficial in easing the development of more complex models.

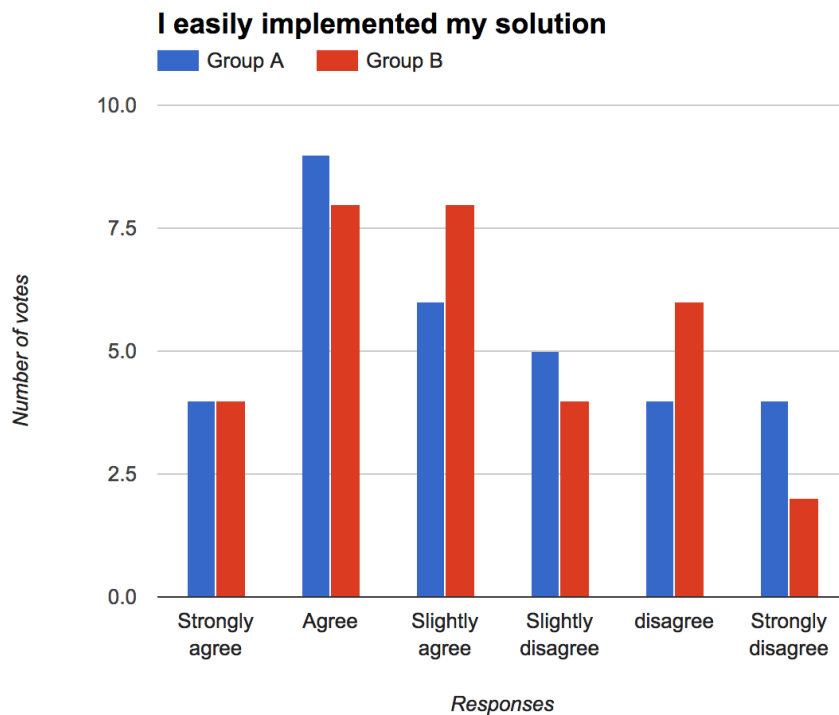


Figure 15. Responses of group A and B to the second survey question.

In the third question, we evaluate to what extent participants feel positive toward solving the problem. Figure 16 shows that group A felt more positive and confident compared to group B.

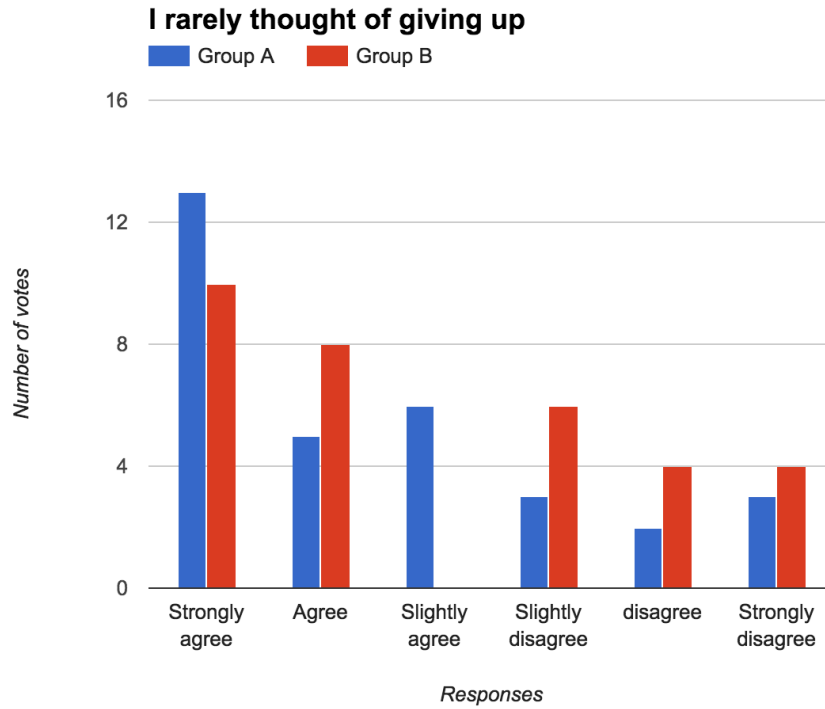


Figure 16. Responses of group A and B to the third survey question.

In the fourth question, we asked the participants if the given time was adequate for them to solve the problem or not. As Figure 17 illustrates, participants in group A had better timing management compared to group B. This is especially interesting when we recall that group B average submission time was shorter than group A. These observations lead us to conclude that group B has faced a struggle in finding a suitable

solution in the given time and has stayed with a straightforward algorithm for solving the problem.

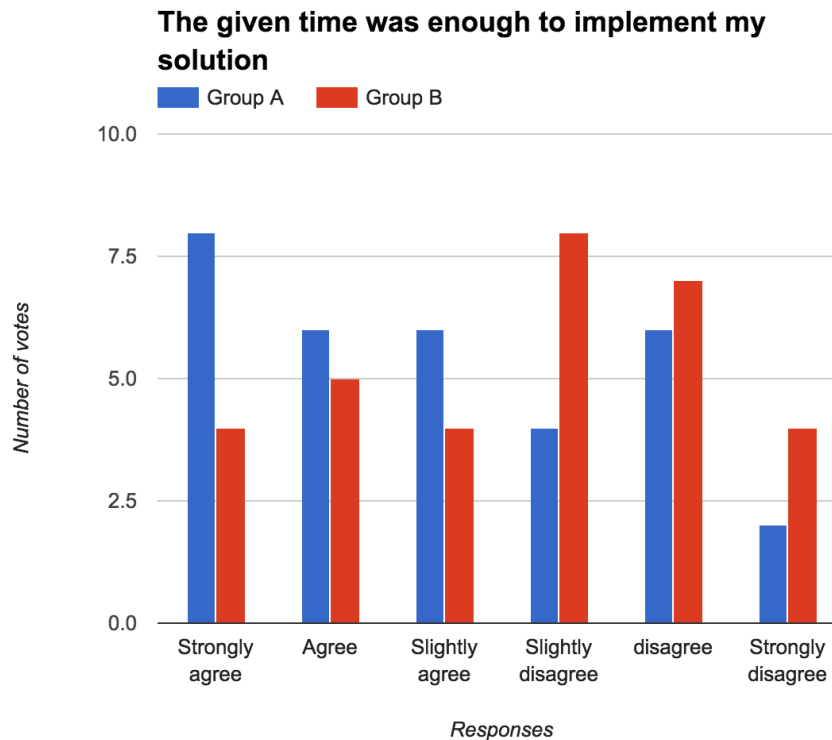


Figure 17. Responses of group A and B to the fourth survey question.

In the fifth question, we asked the participants if they had learned new methods during the experiment or not. As Figure 18 illustrates, many of the participants in group A answer positively to this question, which indicates that they were mostly unfamiliar with classification algorithm before the experiment and learned KNN through the provided tutorials and used it in solving the problem afterward.

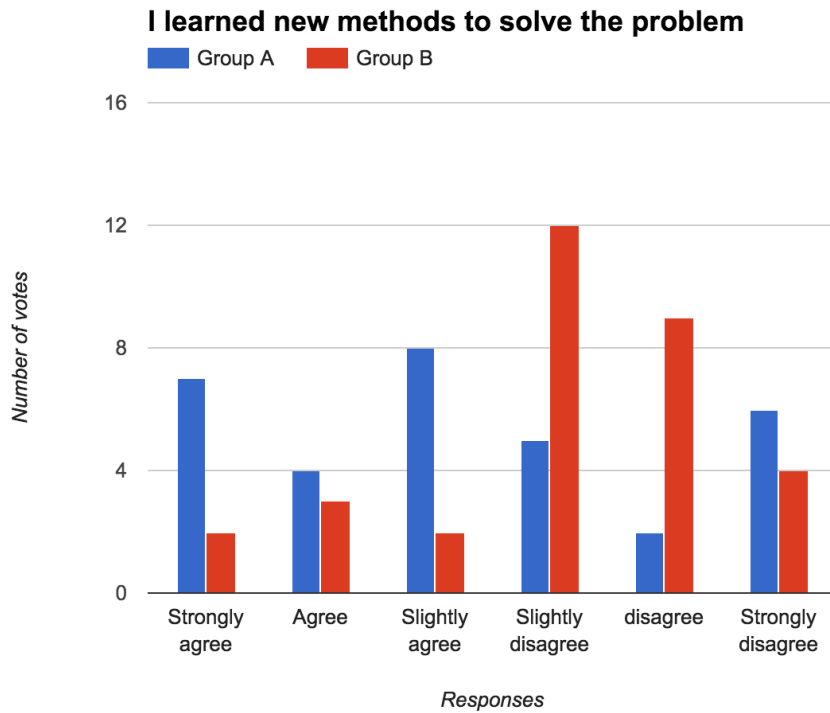


Figure 18. Responses of group A and B to the fifth survey question.

Figure 19 shows to what extent participants find working with our framework easy. Responses show that the majority of participants had no problem in learning the framework and the answers are mostly similar across the two groups.

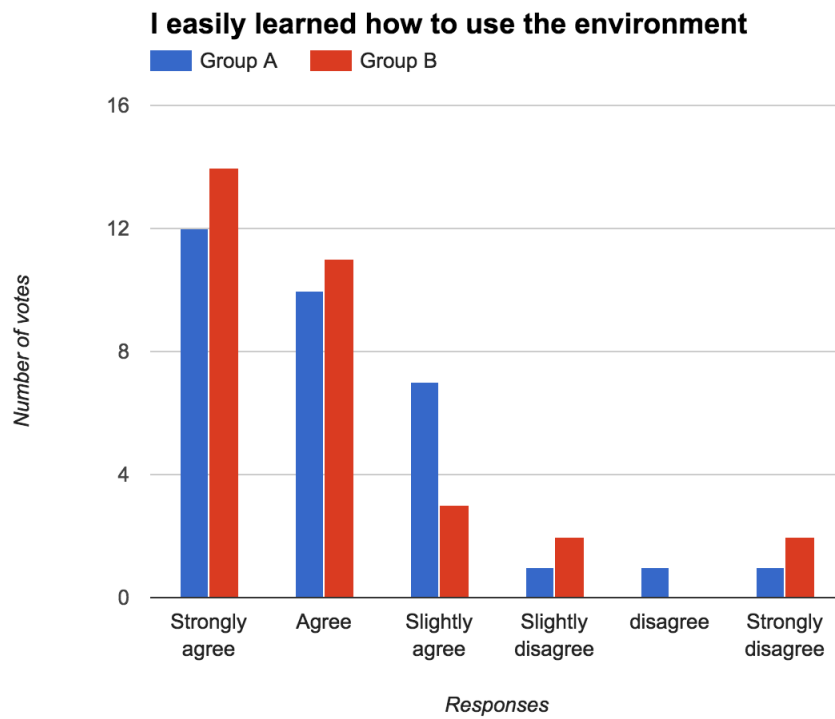


Figure 19. Responses of group A and B to the sixth survey question.

Same results can be observed in the last question (Figure 20), which asks for the ease of use of our framework for implementation, compilation, and run of the programs.

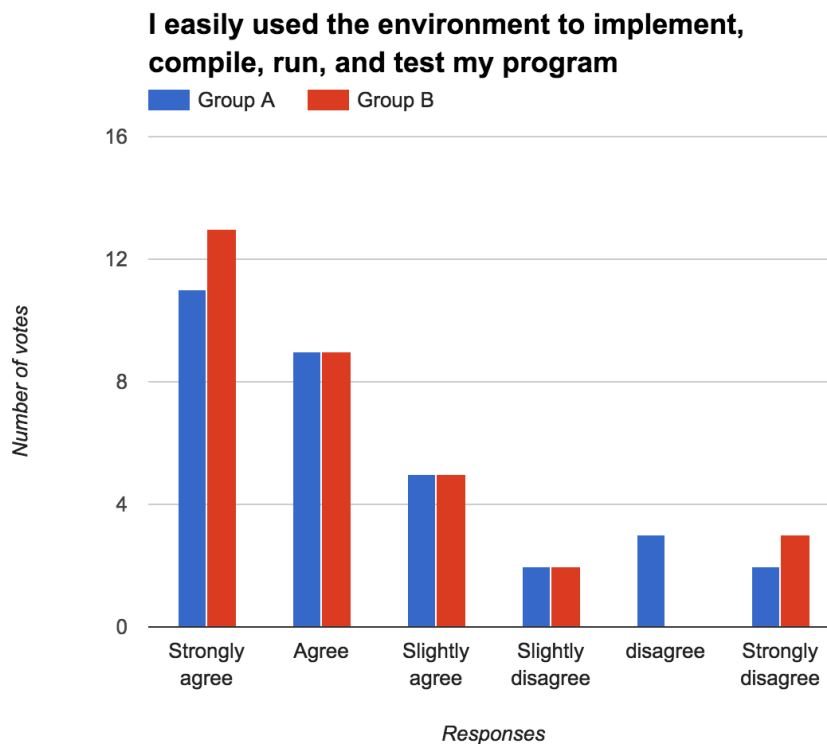


Figure 20. Responses of group A and B to the seventh survey question.

6. Conclusion and Future Direction

In this work, we propose a framework for the development of pattern recognition algorithms for embedded systems. The framework allows developers to employ reference code and an interactive environment to learn and better develop classification algorithms for their specific purpose. We compare advantages and disadvantages of different classification algorithms when employed in the embedded system area and select KNN algorithm to be included in our framework as the main classification algorithm. By conducting an experiment, we demonstrated that using our framework enables developers to come up with better algorithms that lead to higher classification performance. The

feedback collected from developers through a questionnaire also validate benefits of the proposed framework.

A number of possible future directions to improve the current framework involves inclusion of feature extraction techniques. Feature extraction plays a major role in the final performance of classification process and requires expertise. Therefore, providing help in this step also can ease the development process. Furthermore, many embedded systems use time series data, which require specific analysis models to be processed and classified. Therefore, there is also a need to add support for this type of data in the proposed framework. Finally, the current reference code can be expanded to include a wider variety of classification algorithms.

7. References

- [1] Barr, M. (2009). Real men program in C. *Embedded Systems Design*, 22(7), 3.
- [2] Yang, Z., Liu, J., Eric, C., Guan, L. C., & Chin, C. K. (2005, October). An embedded system for speech recognition compression. In *Communications and Information Technology, 2005. ISCIT 2005. IEEE International Symposium on* (Vol. 1, pp. 297-300). IEEE.
- [3] Shim, D., Chung, H., Kim, H. J., & Sastry, S. (2005, August). Autonomous exploration in unknown urban environments for unmanned aerial vehicles. In *AIAA Guidance, Navigation, and Control Conference and Exhibit* (p. 6478).
- [4] Wolf, W., Ozer, B., & Lv, T. (2002). Smart cameras as embedded systems. *computer*, 35(9), 48-53.
- [5] Choudhury, T., Consolvo, S., Harrison, B., Hightower, J., LaMarca, A., LeGrand, L., ... & Klasnja, P. (2008). The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2).

- [6] Okwu, P. I., & Onyeje, I. N. (2013). Ubiquitous Embedded Systems Revolution: Applications and Emerging Trends. *International Journal of Engineering Research and Applications (IJERA)* Vol, 3, 610-616.
- [7] Morganti, E., Angelini, L., Adami, A., Lalanne, D., Lorenzelli, L., & Mugellini, E. (2012). A smart watch with embedded sensors to recognize objects, grasps and forearm gestures. *Procedia Engineering*, 41, 1169-1175.
- [8] Ramya, V., & Palaniappan, B. (2012). Embedded system for Hazardous Gas detection and Alerting. *International Journal of Distributed and Parallel Systems (IJDPS)* Vol, 3, 287-300.
- [9] Shen, W., Gu, J., & Shen, Y. (2005, July). Proposed wall climbing robot with permanent magnetic tracks for inspecting oil tanks. In *Mechatronics and Automation, 2005 IEEE International Conference* (Vol. 4, pp. 2072-2077). IEEE.
- [10] Bramberger, M., Doblander, A., Maier, A., Rinner, B., & Schwabach, H. (2006). Distributed embedded smart cameras for surveillance applications. *computer*, 39(2), 68-75.
- [11] Hata, Y., Kamozaiki, Y., Sawayama, T., Taniguchi, K., & Nakajima, H. (2007, April). A heart pulse monitoring system by air pressure and ultrasonic sensor systems. In *System of Systems Engineering, 2007. SoSE'07. IEEE International Conference on* (pp. 1-5). IEEE.
- [12] White, J. (2005, September). Developing design tools for biological and biomedical applications of micro-and nano-technology. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (pp. 196-200). ACM.
- [13] Ryu, S. W., Kim, C. H., Han, J. W., Kim, C. J., Jung, C., Park, H. G., & Choi, Y. K. (2010). Gold nanoparticle embedded silicon nanowire biosensor for applications of label-free DNA detection. *Biosensors and Bioelectronics*, 25(9), 2182-2185.
- [14] Postolache, O., Girao, P. S., Mendes, J., & Postolache, G. (2009, May). Unobstrusive heart rate and respiratory rate monitor embedded on a wheelchair. In *Medical Measurements and Applications, 2009. MeMeA 2009. IEEE International Workshop on* (pp. 83-88). IEEE.
- [15] Thakare, V. S., Jadhav, S. R., Sayyed, S. G., & Pawar, P. V. (2013). Design of smart traffic light controller using embedded system. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 10(1), 30-33.

- [16] Hughes, D., Greenwood, P., Coulson, G., & Blair, G. (2006). Gridstix: Supporting flood prediction using embedded hardware and next generation grid middleware. In *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a* (pp. 6-pp). IEEE.
- [17] Rowe, A., Rosenberg, C., & Nourbakhsh, I. (2002). A low cost embedded color vision system. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on* (Vol. 1, pp. 208-213). IEEE.
- [18] Shi, G., Chan, C. S., Li, W. J., Leung, K. S., Zou, Y., & Jin, Y. (2009). Mobile human airbag system for fall protection using MEMS sensors and embedded SVM classifier. *IEEE Sensors Journal*, 9(5), 495-503.
- [19] Kong, K., & Tomizuka, M. (2009). A gait monitoring system based on air pressure sensors embedded in a shoe. *IEEE/ASME Transactions on mechatronics*, 14(3), 358-370.
- [20] Lee, Y. D., & Chung, W. Y. (2009). Wireless sensor network based wearable smart shirt for ubiquitous health and activity monitoring. *Sensors and Actuators B: Chemical*, 140(2), 390-395.
- [21] Lara, O. D., & Labrador, M. A. (2013). A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials*, 15(3), 1192-1209.
- [22] Habetha, J. (2006, August). The MyHeart project-fighting cardiovascular diseases by prevention and early diagnosis. In *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE* (pp. 6746-6749). IEEE.
- [23] Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1), 21-27.
- [24] Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (Vol. 398). John Wiley & Sons.
- [25] Theodoridis, S. & Koutroumbas, K. (2008). *Pattern Recognition*. Elsevier Science.
- [26] Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.
- [27] Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press.
- [28] Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural network design* (Vol. 20). Boston: Pws Pub..

- [29] Starzacher, A., & Rinner, B. (2008, December). Evaluating KNN, LDA and QDA Classification for embedded online Feature Fusion. In *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on* (pp. 85-90). IEEE.
- [30] Keogh, E., & Mueen, A. (2017). Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining* (pp. 314-315). Springer, Boston, MA.
- [31] Shi, G., Chan, C. S., Li, W. J., Leung, K. S., Zou, Y., & Jin, Y. (2009). Mobile human airbag system for fall protection using MEMS sensors and embedded SVM classifier. *IEEE Sensors Journal*, 9(5), 495-503.
- [32] Meng, H., Pears, N., & Bailey, C. (2007, June). A human action recognition system for embedded computer vision application. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on* (pp. 1-6). IEEE.
- [33] He, Z. Y., & Jin, L. W. (2008, July). Activity recognition from acceleration data using AR model representation and SVM. In *Machine Learning and Cybernetics, 2008 International Conference on* (Vol. 4, pp. 2245-2250). IEEE.
- [34] Vahid, F., Givargis, T., & Miller, B. (n.d.). RI Tools for Programming Embedded Systems. Retrieved March 3, 2018, from <http://www.cs.ucr.edu/~vahid/pes/RITools/>