FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Computer Vision System for Tactode Programming

Daniel Queirós da Silva

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Jorge Miranda de Sousa

Second Supervisor: Valter Joaquim Ramos Costa

July 30, 2020

© Daniel Queirós da Silva, 2020

Resumo

A programação tangível, quando direcionada à robótica, torna a atividade de programar mais compreensível e direta. Este tipo de programação ajuda no desenvolvimento precoce das capacidades de programação e do pensamento computacional das crianças de uma forma interativa. Desta ideia surgiu o Tactode: um sistema de programação tangível composto por peças tipo *puzzle* e uma aplicação *web* que visa a programação de robôs. Os utilizadores alvo deste sistema são as crianças que, recorrendo às peças, formam um código tátil, tiram uma fotografia ao mesmo e depois podem carregá-lo para a aplicação para, posteriormente, ser testado e executado no robô.

O projeto Tactode encontra-se desenvolvido com base em marcadores ArUco, isto é, cada peça contém um marcador deste tipo que facilita a sua deteção e distinção no código tangível. Posto isto, esta dissertação deu continuidade a este projeto através do desenvolvimento de um sistema de visão computacional capaz de detetar e identificar cada peça em fotografias de códigos Tactode, sem recorrer aos marcadores ArUco.

O trabalho realizado nesta dissertação consistiu na construção de novas peças Tactode recorrendo a impressão 3D com ácido poliláctico (PLA), o desenvolvimento de um sistem de visão computacional, usando *JavaScript* e *OpenCV*, baseado em Histogramas de Gradientes Orientados (HOG) e Máquinas de Vetores de Suporte (SVM) e a construção de quatro *datasets* (dois para treino e dois para teste) para validar experimentalmente o sistema.

O resultado corresponde a um algoritmo baseado em visão e invariável a rotações que recebe uma fotografia de um código Tactode, cujas peças não dispõem de qualquer tipo de marcador adicional e que são detetadas após a fotografia ser segmentada. No fim, é gerada uma matriz composta por variáveis do tipo *string* que correspondem às peças que foram classificadas e estão localizadas na matriz de acordo com as suas posições na fotografia.

O algoritmo foi testado, com recurso a um dos *datasets* de teste, contra outros métodos do estado de arte do domínio de reconhecimento e identificação de objetos, tais como, *Template Matching* e métodos baseados em *Handcrafted Features*, e superou todos eles, atingindo uma exatidão de classificação de **100%** e um tempo médio de execução à volta de um segundo. Para além disso também foi testado com o outro *dataset* de teste, que é bastante maior em número de peças do que o anterior, para que fosse validado de uma forma exaustiva, obtendo uma exatidão de **99.99%**.

Palavras-chave: Ensino de Programação. *Machine Learning*. Robótica. *Supervised Learning*. Visão Computacional.

Sistema de Classificação Computacional ACM:

- Metodologias de computação → Inteligência artificial → Visão computacional → Tarefas de visão computacional → Visão para robótica
- Metodologias de computação → Inteligência artificial → Visão computacional → Problemas de visão computacional → Identificação de objetos

- Metodologias de computação → *Machine learning* → *Supervised learning* → *Supervised learning* por classificação

Abstract

Tangible programming, when applied to robotics, makes programming more understandable and straightforward. This type of programming helps children developing their abilities of programming and computational thinking interactively and at earlier stages of their lives. From this idea came Tactode: a tangible programming system composed by puzzle-like tiles and a web application that aims robot programming. The target users of this system are children who, using the tiles, build a tactile code, take a picture of it and then can upload it to the application to be tested and executed on the robot later.

The Tactode project is developed based on ArUco markers, meaning that each tile have a marker of this type that facilitates its detection and distinction in the tangible code. Therefore, this dissertation continued this project through the development of a computer vision system capable of detecting and identifying each tile in photographed Tactode codes without depending on the ArUco markers.

The work produced in this dissertation consisted on the construction of new Tactode tiles using to 3D printing with polylactic acid (PLA), the development of a computer vision system, using *JavaScript* and *OpenCV*, based on Histograms of Oriented Gradients (HOG) and Support Vector Machines (SVM) and the building of four datasets (two for training and two for testing) to experimentally validate the system.

The result corresponds to a rotation-invariant vision-based algorithm that receives a photograph of a Tactode code, whose tiles do not have any type of additional marker and are detected after segmenting the photograph. At the end, a matrix is generated composed by variables of the type *string*, that correspond to the tiles that were classified and are located in the matrix according to their positions in the photograph.

The algorithm was tested, using one of the test datasets, against other state-of-the-art methods in the domain of object identification and recognition, such as, Template Matching and methods based on Handcrafted Features, and outperformed all of them, reaching a classification accuracy of **100%** and an average execution time around one second. In addition, it was also tested with the other test dataset, which is much larger in number of tiles than the previous one, so that it could be validated exhaustively, achieving a **99.99%** accuracy.

Keywords: Computer Vision. Machine Learning. Programming Education. Robotics. Supervised Learning.

ACM Computing Classification System:

- Computing methodologies → Artificial intelligence → Computer vision → Computer vision tasks → Vision for robotics
- Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems → Object identification

- Computer methodologies → Machine learning → Supervised learning → Supervised learning by classification

Agradecimentos

A realização desta dissertação e o término desta etapa académica só foram possíveis graças às pessoas que me apoiaram, motivaram e que acreditaram em mim. Deixo assim um agradecimento muito especial às seguintes pessoas:

Aos meus orientadores, Professor Doutor Armando Jorge Miranda de Sousa e Doutor Valter Joaquim Ramos Costa, por todo o auxílio e disponibilidade que prestaram ao longo desta dissertação.

À minha namorada por todo o amor e por ter estado sempre lá durante todo o meu percurso académico, nos bons e maus momentos, fazendo-me acreditar que eu iria conseguir.

Aos meus pais e à minha irmã, por todo o carinho e apoio que me deram durante este trajeto.

Aos meus amigos, pelas brincadeiras, convívios, mas sobretudo pela amizade.

Daniel Silva

"Success is walking from failure to failure with no loss of enthusiasm."

Winston Churchill

Contents

Resumo i

Abstract iii

Abbreviations xv

1 Introduction 1
- 1.1 Context and Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Structure of the Document . . . 2

2 Fundamentals and Literature Review 3
- 2.1 Image Processing . . . 3
  - 2.1.1 Colour Spaces . . . 3
  - 2.1.2 Features Extraction . . . 7
  - 2.1.3 Image Segmentation . . . 7
  - 2.1.4 Segmentation Aid . . . 9
  - 2.1.5 OpenCV Contours . . . 14
- 2.2 Image Feature Detection and Description . . . 16
  - 2.2.1 SIFT - Scale Invariant Feature Transform . . . 17
  - 2.2.2 SURF - Speeded-Up Robust Features . . . 18
  - 2.2.3 FAST - Features from Accelerated Segment Test . . . 18
  - 2.2.4 AGAST - Adaptive and Generic Accelerated Segment Test . . . 19
  - 2.2.5 BRIEF - Binary Robust Independent Elementary Features . . . 19
  - 2.2.6 BRISK - Binary Robust Invariant Scalable Keypoints . . . 20
  - 2.2.7 ORB - Oriented FAST and Rotated BRIEF . . . 20
  - 2.2.8 FREAK - Fast Retina Keypoint . . . 20
  - 2.2.9 KAZE . . . 21
  - 2.2.10 AKAZE - Accelerated KAZE . . . 21
  - 2.2.11 HOG - Histogram of Oriented Gradients . . . 21
  - 2.2.12 Comparative Analysis of the Algorithms . . . 22
- 2.3 Machine Learning . . . 22
  - 2.3.1 SVM - Support Vector Machine . . . 23
  - 2.3.2 Deep Learning . . . 25
- 2.4 Evaluation Metrics . . . 26
- 2.5 Development Technologies . . . 27

3 Problem Statement and Solution Proposal **29**
3.1 Problem Definition . . . 29
3.2 Proposed Solution . . . 30
3.2.1 Hardware . . . 30
3.2.2 Software Pipeline . . . 33
3.3 Integration Experiments . . . 47

4 Experimental Validation **49**
4.1 Training Datasets . . . 49
4.2 Test Datasets . . . 50
4.2.1 *Small* Dataset . . . 50
4.2.2 *Big* Dataset . . . 51
4.3 Baseline Classification Methods . . . 52

5 Results **55**
5.1 Preliminary Information . . . 55
5.2 Results of the Tests Executed on the *Small* Dataset . . . 56
5.3 Results of the Tests Executed on the *Big* Dataset . . . 68
5.4 Final Considerations . . . 70

6 Conclusions **71**
6.1 Contributions . . . 71
6.2 Future Work . . . 72

A Datasets Test Codes **73**
A.1 Tactode Codes of the *Small* Dataset . . . 73
A.2 Tactode Codes of the *Big* Dataset . . . 81

B Confusion Matrices of the Tiles **83**
B.1 Confusion Matrices Resulting from the *Small* Test Dataset . . . 83
B.2 Confusion Matrices Resulting from the *Big* Test Dataset . . . 91

References **93**

List of Figures

2.1 RGB colour space three-dimensional representation 4
2.2 HSI colour space 6
2.3 Munsell colour space 7
2.4 Image segmentation techniques 8
2.5 Matrix marker 10
2.6 ARToolKit markers 10
2.7 CyberCode marker 10
2.8 Visual Code marker 10
2.9 ARTag markers 11
2.10 ARToolKitPlus markers 11
2.11 BinARyID marker 11
2.12 ArUco marker 11
2.13 Line Pencil marker 12
2.14 CALTag marker 12
2.15 AprilTag marker 12
2.16 Pi-Tag marker 13
2.17 RUNE-Tag markers 13
2.18 CCTag marker 13
2.19 ChromaTag marker 13
2.20 Examples of Naomarks 14
2.21 Examples of other markers 14
2.22 OpenCV contours on a photographed Tactode tile 15
2.23 Illustrative example of contours hierarchy 16
2.24 The model that YOLO is based on 26

3.1 Examples of original Tactode tiles 29
3.2 Two examples of Tactode codes for taking a square-shaped route 30
3.3 The tile "repeat" without and with coloured printed paper on top 31
3.4 Comparison of dimensions between original tiles and new tiles 31
3.5 Similitude between the "repeat" and "end repeat" tiles with the "forever" and "end forever" tiles 32
3.6 Modifications done on "forever" and "end forever" tiles 32
3.7 New tiles that were built 32
3.8 Block diagram of the software pipeline 33
3.9 Resized Tactode code 34
3.10 Two different images of the same Tactode code 35
3.11 Binary image containing the Tactode code 36
3.12 The main contours 37

3.13 Two images of the same code after the adjustment with extra parts 38
3.14 The most similar contours to the tiles 39
3.15 Positions of the contour centroid and its bounding box center in the tile "lower a" 40
3.16 Two images of extracted tiles from the code with coloured regions 41
3.17 Three images of the tile "flag" representing three different steps of ROI extraction 42
3.18 The separation between the body and corners of tiles 43
3.19 The bounding boxes around the tiles 44
3.20 Two images of the tile "addition" for training the classifier 46
3.21 Resulting matrix with the classification labels 46
3.22 Tactode web application for integration experiments 48

4.1 Three templates of the 32 template set 53

A.1 Code with 11 tiles of the *small* dataset 74
A.2 Code with five tiles of the *small* dataset 75
A.3 Code with four tiles of the *small* dataset 76
A.4 Code with 11 tiles of the *small* dataset 77
A.5 Code with 19 tiles of the *small* dataset 78
A.6 Code with 8 tiles of the *small* dataset 79
A.7 Code with 11 tiles of the *big* dataset 81
A.8 Code with five tiles of the *big* dataset 82

B.1 Confusion matrix of HOG and SVM resulting from the *small* dataset 84
B.2 Confusion matrix of SURF resulting from the *small* dataset 85
B.3 Confusion matrix of SIFT resulting from the *small* dataset 86
B.4 Confusion matrix of BRISK resulting from the *small* dataset 87
B.5 Confusion matrix of ORB resulting from the *small* dataset 88
B.6 Confusion matrix of Template Matching with *CCOEFF* resulting from the *small* dataset 89
B.7 Confusion matrix of Template Matching with *SQDIFF* resulting from the *small* dataset 90
B.8 Confusion matrix of HOG and SVM resulting from the *big* dataset 92

List of Tables

2.1 Examples of contours hierarchy arrays . . . 16
2.2 Summary of the feature detection and description algorithms . . . 22

3.1 Conditions for determining the direction of a tooth of a tile . . . 40
3.2 Possible teeth directions for determining the rotation angle of a code . . . 40

4.1 Specifications of the training dataset for tiles "teeth" detection . . . 49
4.2 Specifications of the training dataset for tile classification . . . 50
4.3 Camera resolutions of the devices used . . . 50
4.4 Computer specifications where the tests were executed . . . 50
4.5 The resolutions present in the *small* dataset and the respective number of images and tiles . . . 51
4.6 The resolutions present in the *big* dataset with the respective rotations and number of images and tiles . . . 52
4.7 Settings of the methods for feature detection and description . . . 53

5.1 Resulting accuracy, ETMV and ETSD from the tests made with various classification methods on the *small* dataset . . . 56
5.2 Resulting correctness level from the tests made with various classification methods on the *small* dataset . . . 57
5.3 Results of the tests on the *small* dataset with HOG and SVM for each resolution . . . 57
5.4 Results of the tests on the *small* dataset with SURF for each resolution . . . 58
5.5 Results of the tests on the *small* dataset with SIFT for each resolution . . . 58
5.6 Results of the tests on the *small* dataset with BRISK for each resolution . . . 58
5.7 Results of the tests on the *small* dataset with ORB for each resolution . . . 59
5.8 Results of the tests on the *small* dataset with Template Matching (*CCOEFF*) for each resolution . . . 59
5.9 Results of the tests on the *small* dataset with Template Matching (*SQDIFF*) for each resolution . . . 59
5.10 Evaluation of the tiles classification for the tests with HOG and SVM on the *small* dataset . . . 60
5.11 Evaluation of the tiles classification for the tests with SURF on the *small* dataset . . . 61
5.12 Evaluation of the tiles classification for the tests with SIFT on the *small* dataset . . . 62
5.13 Evaluation of the tiles classification for the tests with BRISK on the *small* dataset . . . 63
5.14 Evaluation of the tiles classification for the tests with ORB on the *small* dataset . . . 64
5.15 Evaluation of the tiles classification for the tests with Template Matching (*CCOEFF*) on the *small* dataset . . . 65
5.16 Evaluation of the tiles classification for the tests with Template Matching (*SQDIFF*) on the *small* dataset . . . 66

5.17 Resulting accuracy, ETMV and ETSD from the tests made with HOG and SVM on the *big* dataset 68
5.18 Resulting correctness level from the tests made with HOG and SVM on the *big* dataset 68
5.19 Results of the tests on the *big* dataset with HOG and SVM for each resolution . . 68
5.20 Evaluation of the tiles classification for the tests with HOG and SVM on the *big* dataset 69

Abbreviations and Symbols

AGAST	Adaptive and Generic Accelerated Segment Test
AST	Abstract Syntax Tree
BHT	Balanced Histogram Thresholding
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
BST	Binary Search Tree
CIE	Commission International de l'Eclairage
CMY	Cyan Magenta Yellow
CMYK	Cyan Magenta Yellow Key
CPU	Central Processing Unit
CRM	Contour Retrieval Mode
DAG	Directed Acyclic Graph
DoG	Difference of Gaussian
ETMV	Execution Time Mean Value
ETSD	Execution Time Standard Deviation
EVA	Ethylene Vinyl Acetate
FAST	Feature from Accelerated Segment Test
FN	False Negatives
FP	False Positives
FREAK	Fast Retina Keypoint
HOG	Histogram of Oriented Gradients
HSI	Hue Saturation Intensity
JS	JavaScript
npm	Node Package Manager
Nrgb	Normalised RGB
ORB	Oriented FAST and Rotated BRIEF
OS	Operating System
PLA	Polylactic Acid
RAM	Random-Access Memory
RANSAC	Random Sample Consensus
RGB	Red Green Blue
ROI	Region Of Interest
SIFT	Scale Invariant Feature Transform
SURF	Speeded-Up Robust Features
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
YOLO	You Only Look Once

Chapter 1

Introduction

This chapter presents the context and motivation, the objectives and the structure of this dissertation.

1.1 Context and Motivation

Tactode is a project in development whose name comes from Tactile Coding and corresponds to a tangible programming system for robots [1, 2, 3]. Tangible programming when related to robotics, makes the activity of programming more direct and easy to understand [4]. Therefore, Tactode aims at the early development of children's programming skills and computational thinking in a fun and interactive way.

The Tactode programming system is composed by puzzle-like tiles, an web application that can be run at any mobile device such as a computer, tablet or smartphone, a simulated robot and possibly a real one. Therefore, the kids create a tactile programming code using the Tactode tiles, they capture a photograph of the code with the mobile device and then, they can upload it to the web application. In addition, they can test their code using the simulated robot on the application before sending and executing it in a real robot.

The Tactode application is based on the Ionic framework [5], which is a web based multi-platform development framework that makes possible the application to run on various operating systems such as *Android*, *iOS*, *Windows*, *macOS*, and also in the browser. This framework is an open-source UI toolkit that uses web technologies - HTML, CSS and JavaScript - for building applications [5].

Currently, the vision part of the project is developed based on ArUco markers which are part of each Tactode tile and help detect and distinguish each tile in the code.

This dissertation intended to continue the development of the Tactode project by removing the ArUco markers originating smaller and better looking tiles. Thus, in this dissertation new tiles without the ArUco marker were built and was developed a computer vision algorithm capable of recognising the new tiles.

1.2 Objectives

The objective of this dissertation was the design and development of a computer vision system capable of detecting and identifying each Tactode tile by its own content, instead of using the ArUco marker.

1.3 Structure of the Document

In addition to the current introductory chapter, this document is constituted by five more chapters. Chapter 2 contains the state of the art of image processing, feature detection and description in images, machine learning, with more relevance to SVM and deep learning, evaluation metrics and specifications of development technologies. In Chapter 3 is presented the problem at hand, the proposed solution and integration experiments. The Chapter 4 describes the testing conditions and the created datasets, and Chapter 5 presents the results obtained and their interpretation. Lastly, Chapter 6 ends this document concluding about the work developed, the results that were accomplished, the contributions made with this dissertation and some domains that can be explored and traduced into future work.

Chapter 2

Fundamentals and Literature Review

In this chapter is shown the state of the art of image processing, specifically colour space, segmentation, fiducial markers and a brief presentation of *OpenCV* contours. The domains of the detection and description of image features, and machine learning with SVM and a well-known deep learning method are also presented. Lastly, this chapter ends with a revision over models evaluation metrics and some development technologies that were used during this dissertation.

2.1 Image Processing

In this section are addressed themes that belong to the domain of image processing which corresponds to the digital processing of an image by an algorithm.

2.1.1 Colour Spaces

Colour spaces are representations that specify colours through a mathematical model. These colour models are important in image processing since they allow a deeper understanding of colour digital images.

2.1.1.1 RGB

RGB is a colour space whose name comes from the initials of the three primary colours: Red, Green and Blue. This model can be represented in a three-dimensional cube (Figure 2.1) where each point coordinates correspond to the component values of red, green and blue. Also, this model is appropriate to display colour, but is inadequate for colour analysis due to the high correlation between the components, i.e., the three components (red, green and blue) change according to intensity changes [6].

A normalised version of RGB (Nrgb), that is less sensitive to illumination changes, is defined by (2.1).

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B} \tag{2.1}$$

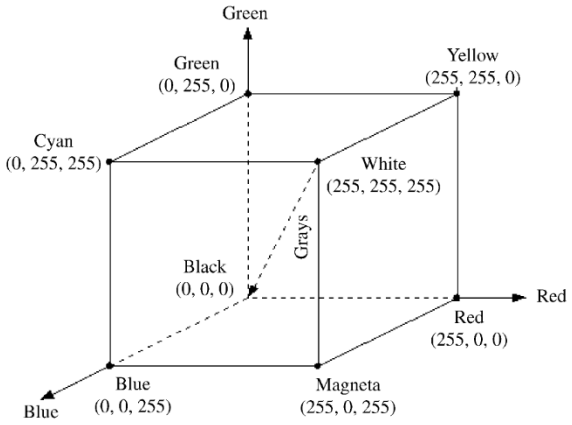


Figure 2.1: RGB colour space three-dimensional representation [6].

2.1.1.2 YIQ

YIQ is colour space used for colour TV broadcasting and is attained by a linear transformation of RGB defined by (2.2), where $0 \leq R \leq 1, 0 \leq G \leq 1, 0 \leq B \leq 1$.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.253 & -0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.2}$$

This model is formed by three components: the Y measures the colour luminance and the I together with Q characterise the image hue and saturation. Moreover, this colour space does not appear to have correlation between the red, green and blue components of an image [6].

2.1.1.3 YUV

YUV is another colour space used for colour TV broadcasting. The Y component represents the luminance and the U and V components represent the chrominance [7]. This model is obtained through RGB following (2.3), where $0 \leq R \leq 1, 0 \leq G \leq 1, 0 \leq B \leq 1$.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.3}$$

2.1.1.4 YCbCr

YCbCr is a colour space used for digital video and is composed by a luminance component Y and, the Cb and Cr components are the chroma components which represents the blue and red differences [7]. To obtain this model a transformation is made from RGB and is defined by (2.4), where $0 \leq R \leq 1, 0 \leq G \leq 1, 0 \leq B \leq 1$.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.4}$$

2.1.1.5 CMY(K)

CMY(K) is a colour model whose name comes from the subtractive primary colours: cyan, magenta and yellow (and black). This model is mostly used in printers [7]. The RGB to CMY transformation is defined by (2.5).

$$C = 1 - R, \quad M = 1 - G, \quad Y = 1 - B \tag{2.5}$$

To obtain CMYK from CMY it is required to perform the transformation specified in (2.6).

$$K = min(C, M, Y), \quad C = \frac{C - K}{1 - K}, \quad M = \frac{M - K}{1 - K}, \quad Y = \frac{Y - K}{1 - K} \tag{2.6}$$

2.1.1.6 HSI

HSI is a colour system that perceives an image by its colour and intensity information [6]. The components of this model are hue (H), saturation (S) and intensity (I). The colour information is based on the values of hue and saturation and the intensity information (image brightness) is based on the amount of light. This model is geometrically represented in Figure 2.2 and its components are computed applying the transformations shown in (2.7).

$$H = arctan\left(\frac{\sqrt{3}(G - B)}{(R - G) + (R - B)}\right), \quad I = \frac{R + G + B}{3}, \quad S = 1 - \frac{min(R, G, B)}{I} \tag{2.7}$$

2.1.1.7 CIE

CIE (Commission International de l'Eclairage) is a colour system that has three components denoted as X, Y and Z through which any colour can be defined [6]. The values of the components are calculated by the transformation defined in (2.8):

$$\begin{bmatrix} X \\ Y \\ X \end{bmatrix} = \begin{bmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.8}$$

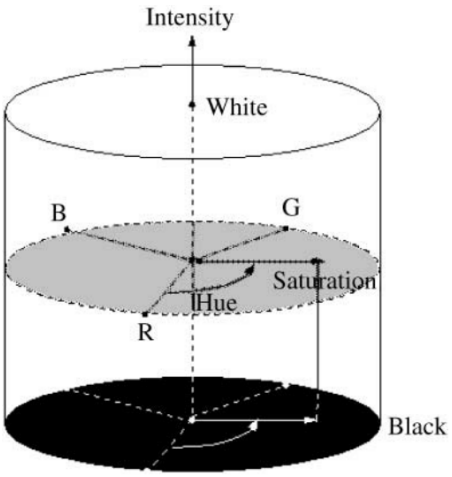


Figure 2.2: HSI colour space [6].

From XYZ coordinates can be obtained other CIE spaces such as CIE (L*a*b*) and CIE (L*u*v*) which share the value L*, that determines colour lightness, and are both obtained through transformations from RGB space.

The CIE (L*a*b*) is defined by (2.9), where $Y/Y_0 > 0.01$, $X/X_0 > 0.01$, and $Z/Z_0 > 0.01$. The X_0, Y_0, Z_0 are X, Y, Z values for standard white.

$$L^* = 116\left(\sqrt[3]{\frac{Y}{Y_0}}\right) - 16, \quad a^* = 500\left(\sqrt[3]{\frac{X}{X_0}} - \sqrt[3]{\frac{Y}{Y_0}}\right), \quad b^* = 200\left(\sqrt[3]{\frac{Y}{Y_0}} - \sqrt[3]{\frac{Z}{Z_0}}\right) \tag{2.9}$$

The CIE (L*u*v*) is defined by (2.10), where $Y/Y_0 > 0.01$, Y_0, u_0 and v_0 are standard white values, and $u' = 4X/(X+15Y+3Z)$, $v' = 6Y/(X+15Y+3Z)$.

$$L^* = 116\left(\sqrt[3]{\frac{Y}{Y_0}}\right) - 16, \quad u^* = 13L^*(u' - u_0), \quad v^* = 13L^*(v' - v_0) \tag{2.10}$$

2.1.1.8 Munsell

Munsell is a colour space composed by three components for colour specification: Munsell Hue, Value and Chroma [6]. There are five main hues in this model, which are red (R), yellow (Y), green (G), blue (B) and purple (P). These hues when combined form five half-way hues: YR, GY, BG, PB and RP. This colour system is presented in Figure 2.3. The hue circle is divided into 100 equal parts containing the five major and half-way hues equally spaced [6]. The Munsell value defined the lightness of a colour, specifying black as 0 and white as 10. The Munsell chroma is a measure for colour saturation.

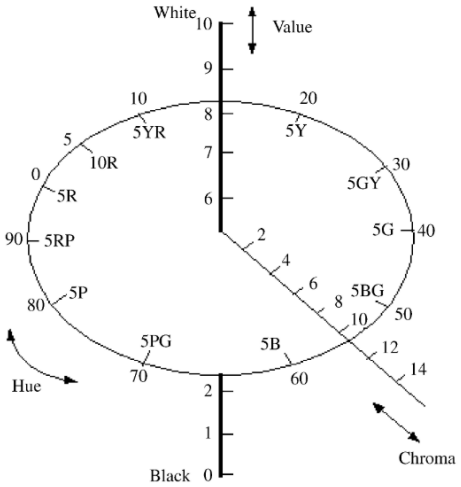


Figure 2.3: Munsell colour space [6].

2.1.2 Features Extraction

In this section are presented two methods for extracting features from images: Hough Transform and Template Matching.

Hough Transform Hough Transform is a feature extraction method whose purpose is to find shapes in images. This is a patented technique [8] that is widely used to detect circumferences, ellipses and lines in images. This technique maps the points of an image into an accumulator space to detect the existence of a certain pattern within the image.

Template Matching Template Matching consists in computing a similarity measure between each template and the target image. This procedure results on the classification of the target image with the class of the template with the highest similarity or the location of the latter in the image [9].

2.1.3 Image Segmentation

Image Segmentation is a crucial step in image processing. The segmentation methods are used for dividing an image into different regions of interest and are based on two properties: discontinuity and similarity [10]. There are several image segmentation techniques and Figure 2.4 presents some that will be discussed in this section.

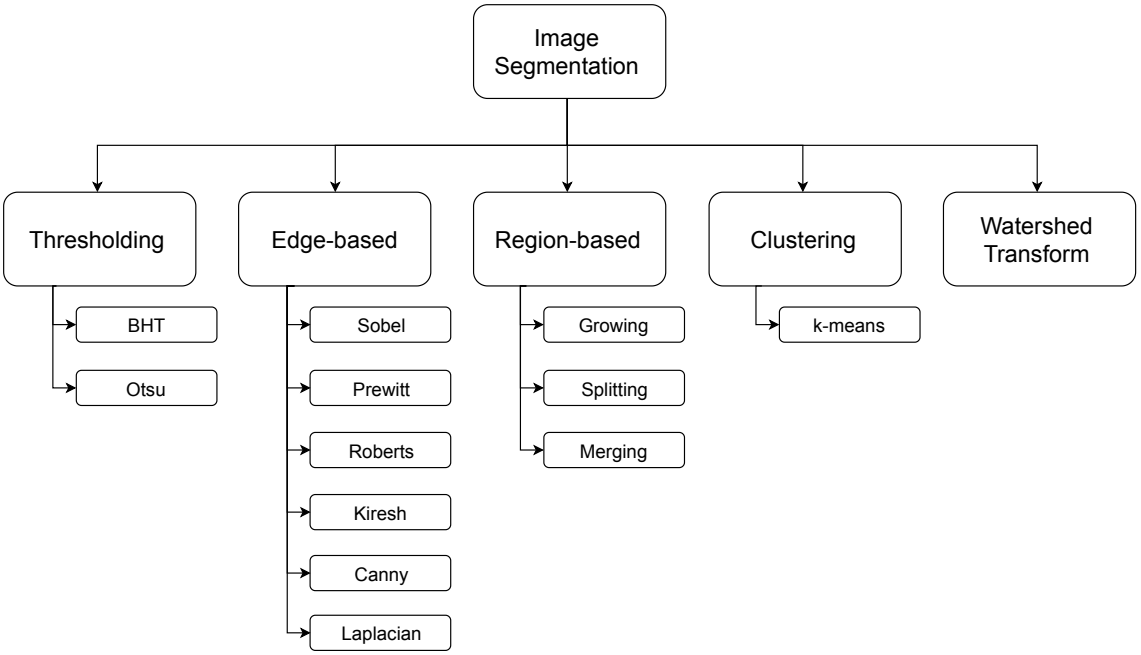


Figure 2.4: Image segmentation techniques.

2.1.3.1 Histogram Thresholding

Histogram Thresholding is an image segmentation technique that separates the histogram of an image into peaks corresponding to certain regions and obtains, between two different peaks, a threshold value that corresponds to the valley [6].

Several methods are used to perform thresholding segmentation based on histograms such as:

- **Balanced Histogram Thresholding (BHT) method:** computes the optimum threshold level that separates the image histogram in two classes: foreground and background. BHT determines the weight of the histogram, verifies which class is heavier and removes weight from the latter until it becomes the lighter, in order to balance the histogram [11].

- **Otsu's method:** determines a threshold that divide pixels in two classes: background and foreground. This threshold is determined by maximising the variance between classes [12].

2.1.3.2 Edge-based

Edge detection-based segmentation is an image segmentation method to discover boundaries based on the detection of discontinuities in inherent image properties such as texture, colour and gray levels [10]. The edge detection techniques are divided into two categories: sequential and parallel. Sequential detection indicates a dependence relationship between the result at a point and the results at previous points. On the other hand, parallel detection presents an independence relationship regarding the decision of a set of points being or not on an edge and other sets of points

whether lie or not on an edge [6]. Some of the currently known edge detectors are: Sobel, Prewitt, Roberts, Kiresh, Canny and Laplacian.

2.1.3.3 Region-based

Region-based approaches towards image segmentation such as region growing, region splitting, region merging and their combination, seek to arrange pixels into regions [6]. The region growing method starts by selecting a reference region which is expanded including its neighbours that satisfy a similarity criterion, forming a cluster. This process is repeated until all the pixels in the image have been covered. Region splitting is an approach in which the initial seed region corresponds to the whole image. Then, if the seed region is not homogeneous, it is split in four sub-regions that in turn become new seed regions [6]. This procedure is repeated until all sub-regions present uniformity. The region merging can be combined with region growing or region splitting to perform a merge of the identical regions.

2.1.3.4 Clustering

Clustering, in image processing, refers to the arrangement of image content into groups or classes. The most used clustering method is k-means that divides an image in k clusters. The center of some clusters are placed in certain locations, each pixel is attached to the cluster with nearest center, then the center of each cluster is re-calculated by the average of the pixels within the cluster. The two previous steps are repeated until there is no swap of pixels between the clusters [13].

2.1.3.5 Watershed Transform

Watershed transform is a segmentation technique that considers the image as a topographic map in which the height is represented by the intensity of each pixel [14]. This technique aims to find regions with extremely proximity in such a way that their edges touch. Analogously to the operation of this method, assuming that catchment basins are being filled with water, when they are both flooded, the areas where the floodwater from different basins meet are identified and in these regions are formed pixel barriers partitioning the image.

2.1.4 Segmentation Aid

This section addresses the current state of the art in the field of detection markers by presenting existing markers and also others that were considered in this dissertation as an aiding method for segmenting an image.

Matrix Matrix is a two-dimensional marker and is based on a square shaped barcode which can identify 2^{16} different objects [15]. This tag system is used to detect objects and estimate their position and orientation. A marker of this type is shown in Figure 2.5.



Figure 2.5: Matrix marker [15].

ARToolKit ARToolKit is a toolkit that contains a fiducial marker system [16]. The markers are composed by a white background, a black border and an interior with a greyscale image which is compared to a set of prototypes by interrelationship. As an example, is shown a group of these markers in Figure 2.6.



Figure 2.6: ARToolKit markers [16].

CyberCode CyberCode is a system of visual tagging based on barcode technology [17]. An example of one of this type of markers is presented in Figure 2.7. The CyberCode can encode 24 or 48 bits of data excluding error correction bits and, as in other barcode-based systems, the information is encrypted following a two-dimensional model.

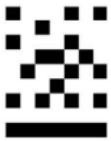


Figure 2.7: CyberCode marker [18].

Visual Code Visual Code is a visual tagging system based on two-dimensional barcodes. A marker of this tagging system is presented in Figure 2.8. In comparison to CyberCode, this code can encode 76 bit-information (83 without error detection) and has a second guide bar, being more reliable in recognising codes at greater inclinations [19].



Figure 2.8: Visual Code marker [18].

ARTag ARTag is a fiducial marker system that was inspired by ARToolKit [18]. This approach uses digital encoding on square markers being based on two-dimensional barcodes. An improvement relatively to ARToolKit is that, ARTag presents immunity to partial occlusion and

lower sensitivity to lighting variations [16]. A few ARTag markers are presented as example in Figure 2.9.

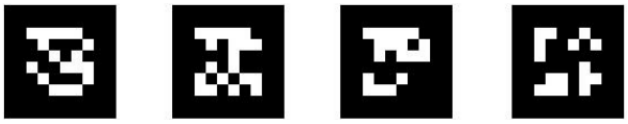


Figure 2.9: ARTag markers [16].

ARToolKitPlus ARToolKitPlus is a marker tracking system and is presented in [20] as a successor to ARToolKit. The main difference is in terms of performance on mobile devices such as smartphones. A few examples of markers of this system are shown in Figure 2.10. These markers are based on binary patterns in a similar way to ARTag. Currently, ARToolKitPlus provides up to 4096 different patterns.



Figure 2.10: ARToolKitPlus markers [16].

BinARyID BinARyID is a marker generation and management system based on markers represented by bitstrings. This system currently is an extension of ARToolKit but is easily adaptable to other marker tracking systems. In fact, it is designed as a small and self-contained extension for existing tracking systems. One example marker of this system is shown in Figure 2.11. The main advantages of this approach are its fast integration, portability and ease of use [21].



Figure 2.11: BinARyID marker [18].

ArUco ArUco is a fiducial marker based on two-dimensional barcodes. It is a well known marker due to its implementation in the open-source library *OpenCV*, being one of the most popular marker detection systems. This system is able to detect and correct errors of the codes [22]. Figure 2.12, shows an example of an ArUco marker.



Figure 2.12: ArUco marker [22].

Line Pencil Line Pencil is an optical tracking system that consists of line fiducials. This approach is based on the recognition of patterns composed by *pencils* which are four lines that intersect at a point [23]. In addition, this strategy reduces occlusion problems. An example of a Line Pencil is demonstrated in Figure 2.13.



Figure 2.13: Line Pencil marker [24].

CALTag CALTag is a fiducial marker system whose name comes from *CALibration Tags* [25]. This system uses checkerboard-based markers which are composed by inner square markers. This type of markers increase the number of corners to perform an improved estimation of the camera pose. An example of a CALTag marker is presented in Figure 2.14.



Figure 2.14: CALTag marker [25].

AprilTag AprilTag is a visual fiducial system that uses a two-dimensional barcode based tag [26]. This system presents a coding pattern stronger than, for example ARTag and ARToolKitPlus, encoding up to 4164 different codes with an improvement in minimum Hamming distance. An example of one these markers is shown in Figure 2.15.



Figure 2.15: AprilTag marker [27].

Pi-Tag Pi-Tag is a marker whose name comes from *Projective invariant Tag* [28]. It is based on a simple design being a square of 12 dots which are placed on the sides resulting in four dots per side and the corner dots being shared. This marker presents two different configurations of the dots, repeating each in two adjacent sides. In Figure 2.16 is presented an example of a Pi-Tag marker and can be seen that the top and left sides share the same configuration as well as the bottom and right sides do.

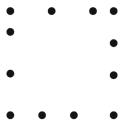


Figure 2.16: Pi-Tag marker [28].

RUNE-Tag RUNE-Tag is a tag constructed by the division of a disk in equally distributed *sectors*. Each sector is divided into *levels* which are concentric rings. Each pair formed by a sector and a level represents a *slot* where is placed a *dot*. Therefore, each dot is a circular feature and its radius is proportional to the radius of the level where the dot is placed [29]. This design facilitates to easily locate and attach information to the marker. In Figure 2.17, is shown two examples of RUNE-Tag markers.



Figure 2.17: RUNE-Tag markers [29].

CCTag CCTag is a circular fiducial marker that provides robustness to blur due to the set of rings used. The ring width encodes the marker code [30, 31]. An example marker of this type is presented in Figure 2.18.



Figure 2.18: CCTag marker [27].

ChromaTag ChromaTag is a colour fiducial marker whose detection algorithm is faster than other markers [27]. Each marker is composed by red and green regions due to their ease of detection and false positives limitation. The code embedded in the marker is formed through the utilisation of two shades of red and green. The surrounding black and white border provides higher accuracy concerning the localisation of the markers. An example of these markers is shown in Figure 2.19.

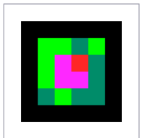


Figure 2.19: ChromaTag marker [27].

Naomark Naomark is a detection marker commonly used in robotics and is composed by black circles and triangle fans centered at the center of the circle as can be seen in Figure 2.20. The particular location of the fans is useful to distinguish each Naomark [32].

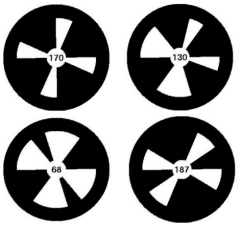


Figure 2.20: Examples of Naomarks [32].

Other markers In addition to the markers mentioned in the above sections, there are others, such as the ones presented in Figure 2.21, that can be considered useful due to being composed of circumferences and straight lines. The last are important characteristics because they facilitate the detection of these markers by the Hough transform that is capable of accurately identifying such morphological characteristics as it was mentioned in Section 2.1.2.



Figure 2.21: Examples of other markers (from the left to the right): Concentric square marker, concentric circle marker and circular marker with inner lines.

2.1.5 OpenCV Contours

Contours are curves that connect points that have similar properties such as colour and intensity, resulting in an outline that surrounds a certain object. This method is very useful for object detection and recognition, and demonstrates more accuracy when applied on binary images (black and white images) [1] [33]. Figure 2.22 shows the contours found in an image.

The implementation of contours in *OpenCV* have a lot of useful methods and properties such as, getting a contour area, perimeter, its moments - spatial, central and central normalised - and the respective bounding box that can be utilised for extracting a Region Of Interest (ROI).

One particular feature of contours is the parent-child relationship that they can have also know as hierarchy. Sometimes when finding contours in an image, some of the obtained contours can be inside others - nested contours - being that, the outer ones are the "parent" and the inner ones the "children". This relationship between contours defines their hierarchy [2] [33].

[1] OpenCV. Contours: Getting Started. Available at https://docs.opencv.org/4.4.0/d5/daa/tutorial_js_contours_begin.html. Last accessed 2020-07-05.

[2] OpenCV. Contours Hierarchy. Available at https://docs.opencv.org/4.4.0/da/d0a/tutorial_js_contours_hierarchy.html. Last accessed 2020-07-05.



Figure 2.22: OpenCV contours on a photographed Tactode tile.

In Figure 2.23, there are numbered shapes from zero to four, where **2** and **2a** indicate the external and internal contours of the respective rectangle. According to the image, contours 0, 1 and 2 are in hierarchy level 0, contour 2a is in the next hierarchy level (level 1) and is considered to be a child of contour 2 (or contour 2 is its parent). Lastly, contours 3 and 4 are in the last hierarchy level (level 2) and they are both children of contour 2a.

In *OpenCV*, the contours hierarchy is represented by an array with four values: **[Next, Previous, First_Child, Parent]**. **Next** indicates the next contour at the same hierarchical level, **Previous** corresponds to the previous contour at the same hierarchical level, **First_Child** represents the first child contour and **Parent** is the parent contour [2] [33]. The values in the array are the indexes of the respective contours.

The way hierarchy is calculated over the contours is specified by the Contour Retrieval Mode (CRM). *OpenCV* offers four types of CRM [2] [33]:

1. **LIST:** simply retrieves all contour without caring with their relationships with each other. The notion of parent and child does not exist with this mode.
2. **EXTERNAL:** only the outer contours are retrieved and the child contours are ignored.
3. **CCOMP:** all contours are retrieved and arranged on a two-level hierarchy. The outer contours are placed on level 1 and the contours inside them are placed in level 2. If any inner contour (level 2) have any other inside itself, then first is placed in level 1 and the latter goes to level 2, and so on.
4. **TREE:** all contours are retrieved and a full family hierarchy list is built.

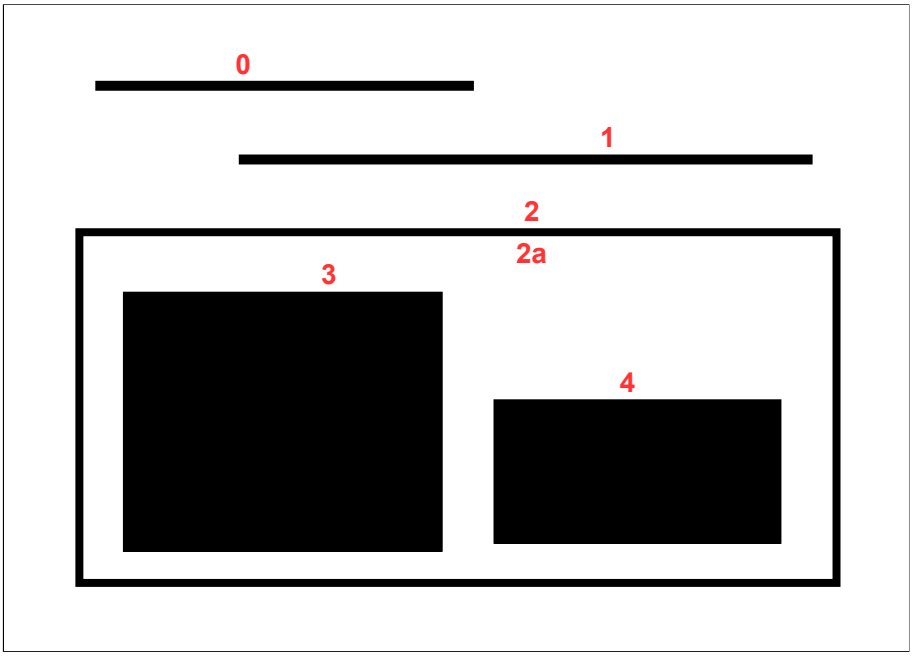


Figure 2.23: Illustrative example of contours hierarchy.

The last mode (TREE) is very interesting because with the complete hierarchy obtained by it, filtering operations to find relevant contours can be accomplished. As a matter of fact, this mode is exploited and will be highlighted later in this dissertation. Therefore, as an example of its use, Table 2.1 presents the hierarchy arrays of the contours shown in Figure 2.23. It is important to notice that when any value of the array equals to -1 means that, the value does not exist.

Table 2.1: Examples of contours hierarchy arrays.

Contour	**Index**	**Hierarchy array**
0	0	[1, -1, -1, -1]
1	1	[2, 0, -1, -1]
2	2	[-1, 1, 3, -1]
2a	3	[-1, -1, 4, 2]
3	4	[5, -1, -1, 3]
4	5	[-1, 4, -1, 3]

A quick analysis of the above table allows concluding that in the case of wanting, for instance only the child contours of the contour *2a*, it would be only needed to check the equality of the last element of the hierarchy arrays of all other contours with the index of the contour *2a*.

2.2 Image Feature Detection and Description

In this section is presented an overview of the algorithms and methods studied regarding the detection, description and matching of features in images as way of recognising objects in them.

2.2.1 SIFT - Scale Invariant Feature Transform

SIFT is a feature detector and descriptor [34, 35]. This algorithm is mainly divided into four stages:

1. Scale-space Extrema Detection: the scale-space of an image is produced from the convolution of a Gaussian blur at different scales with an input image [35]. The scale-space is composed by several octaves and for each of them, the input image is continually convolved with Gaussians resulting on a set of blurred images. The scale difference from one octave to another is by a factor of two. The blurred images are convolved using the Difference of Gaussian (DoG) function, providing efficiency for the detection of keypoints. A keypoint is an image feature that is distinct and can be robustly detected. To detect the extrema, a comparison is made between each pixel of a DoG image, its eight neighbours and the nine neighbours of the DoG images with a scale above and below. The pixel is selected as a keypoint candidate if it is smaller or larger than all of its neighbours.

2. Keypoint Localisation: in this stage, the candidate keypoints obtained from the previous step are evaluated according to their stability and matching. Some of the candidates can have low contrast or can be localised along an edge. In both of the cases, the candidate keypoint must be eliminated and not considered as a keypoint. The quadratic Taylor expansion of the DoG scale-space function is used in order to obtain a more accurate location of the candidate keypoints [35]. If the intensity of a candidate is lower than 0.03, it is rejected. Since the DoG function generates strong responses along edges, the principal curvature is computed using a Hessian matrix. If the candidates have a ratio between the curvatures greater than a predefined value r, they are eliminated.

3. Orientation Assignment: the objective of this step is to assign rotation invariance to the keypoints. The gradient magnitude and orientation are computed for each keypoint obtained in the preceding step. Then an histogram is formed from the gradient orientations of neighbours within a region around the keypoint. The histogram covers the 360 degree range with 36 bins [35]. The weight of each neighbour added to the histogram is measured by its gradient magnitude and a Gaussian-weighted circular window. Existing peaks in the histogram are dominant directions. Multiple keypoints are created at the same location, with the same scale and different orientation for locations where exist multiple peaks.

4. Keypoint Descriptor: at this stage the keypoints are invariant to location, scale and orientation. In this step, the descriptor for each keypoint is constructed to provide invariance to viewpoint and illumination changes and empower match keypoints. The descriptor is attained by computing the gradient magnitude and orientation at each neighbour point in a region around the keypoint location. The weight of these is calculated using a Gaussian window. These neighbour points are then assembled into histograms of orientation and compiled into keypoint descriptors that are invariant to scale, rotation and restricted affine variations [35].

2.2.2 SURF - Speeded-Up Robust Features

SURF is a method for feature detection and description [36]. The feature detector developed for this method is based on an approximation of the Hessian matrix associated with the utilisation of integral images that gained more recognition with the work done by Viola and Jones [37]. These factors help reducing the computational cost thus enhancing the detector performance. The entry of an integral image at a certain location x is the sum of all pixels from the input image inside a rectangular region composed by the origin and x. The Hessian matrix detects groups of keypoints at locations where its determinant is maximum. In addition, the determinant of the Hessian matrix is used to select the scale. This matrix is formed by convolved elements of the Gaussian second order derivative at a certain scale and location. In order to reduce the time consuming of the Gaussian second order derivative computation, an approximation was made based on box filters that along with the integral images improves SURF performance.

The scale-space analysis is done by implementing images pyramid where the images are repeatedly blurred with a Gaussian and sub-sampled until the top of the pyramid is achieved. As consequence of using box filters and integral images, the scale-space analysis is done by up-scaling the filter size instead of reducing the image size. As was aforementioned in SIFT, the scale-space is divided by octaves. In each octave, the filter size is twice larger as well as the sampling intervals for extracting the keypoints. This approach entails a reduction on the computation time and an increase in the accuracy. To localise the keypoints, a variant of a non-maximum suppression is utilised in $3 \times 3 \times 3$ neighbourhood.

The keypoint description comprises an orientation assignment followed by keypoints extraction. Concerning the orientation assignment, the Haar wavelet responses are calculated in the x and y directions within a circular region with radius of $6s$, where s is the corresponding scale at which the keypoint was identified. After having those responses, the vertical and horizontal components of them are summed resulting on a orientation vector. The longest vector found of this type represents the main orientation of the keypoint. For the extraction step, a square region centered around the keypoint and with the orientation obtained from the previous step is constructed and, then it is separated into smaller and square 4×4 sub-regions. For each sub-region, the Haar wavelet responses are computed with equally spaced 5×5 samples. The horizontal and vertical responses are weighted with a Gaussian to increase the stability related to geometric deformations and localisation errors [36]. Then, these responses and its absolute values are summed for each sub-region bringing information about the intensity changes polarity. Hence, each sub-region has a vector with four dimensions resulting on a descriptor vector for all 4×4 sub-regions with a length of 64. The resulting feature descriptor presents invariance to scale and rotation, but it is less invariant to affine changes.

2.2.3 FAST - Features from Accelerated Segment Test

FAST is a feature detector used to identify keypoints in an image [38]. This method operates by firstly considering a candidate pixel p with intensity Ip to be a keypoint, then it is defined a

threshold t as well as a circle around p with radius r. In order to p be considered as a keypoint, there must exist at least n pixels inside the circle with intensity levels above $Ip + t$ or below $Ip - t$: p is not a keypoint if at least m pixels inside the circle do not have intensity levels below or above $Ip + t$; else if at least m pixels have intensity levels above or below $Ip + t$, then the remaining pixels may be verified to check if they accomplish this condition. This procedure is repeated for all pixels. A machine learning approach is used to address some weaknesses and begins by detecting candidate keypoints from a set of images. For each pixel, all the 16 locations within the circle around it are tested and, for each location the pixel at that position can have one of three states: it can be darker, similar or brighter than the nucleus. The creation of a decision tree that can classify keypoints identified in the training set and manifest the rules of the detector allows using it as a reliable keypoint detector. To address a problem defined as the detection of multiple features being adjacent to one another, a non-maximal suppression is applied removing keypoints which have an adjacent keypoint with a higher score [38].

2.2.4 AGAST - Adaptive and Generic Accelerated Segment Test

AGAST is a feature detector and corresponds to a more generic and efficient version of FAST [39]. This method starts by configuring the space for a Binary Search Tree (BST). At this step is defined how a pixel is evaluated regarding its intensity level relatively to the nucleus, and its preceding state. The possible states are: darker, not darker, similar, not brighter and brighter. The next step was to build the optimal decision tree using an algorithm that starts from the root and other nodes are formed sequentially by evaluating a certain pixel. The configuration space is explored until a leaf is found. In AGAST, an optimised solution was achieved with respect to the automatic and dynamic adaptation of the algorithm to an arbitrary scene or to any image section. This is done by using adaptive tree switching that consists on the switch between decision trees when, at the end of each decision path, the pixel configuration of the leaf met or not a certain keypoint criterion - if the pixel is or is not a keypoint. Based on the previous, is performed a jump to an appropriate specialised tree. The specialisation of each tree could be for homogeneous or structured regions based on a small or large value of the probability of a pixel state to be similar to the nucleus. Therefore, exists only one case where AGAST would be efficiently worse than FAST which is if the environment switched from homogeneous to structured and so on at consecutive pixels [39]. In addition, this algorithm needs no learning to fulfil its requirements, while in FAST a learning step is needed.

2.2.5 BRIEF - Binary Robust Independent Elementary Features

BRIEF is a feature descriptor that represents an image patch as binary strings [40]. After smoothing the image patch, its pixel intensities are compared on a pair of points, being these defined as binary tests thus resulting on a bit vector. The resulting descriptor is not invariant to scale and rotation. Lastly, to compare and match the resulting bit strings is used the Hamming distance.

2.2.6 BRISK - Binary Robust Invariant Scalable Keypoints

BRISK is a method used for keypoint detection, description and matching [41]. This algorithm starts by the keypoint detection in the scale-space where the keypoints are detected using the AGAST method. Using the FAST score *s*, the previously obtained keypoints are filtered and the search for maxima is done in the scale-space pyramid, thus achieving scale invariance.

In the keypoint description step, the robustness of the keypoint descriptor is fulfilled by determining the direction of each keypoint in order to have orientation-normalised descriptors and also to obtain rotation invariance. Then, the descriptor is constructed by aggregating the results obtained from brightness comparison tests performed in pairs. As in BRIEF, the Hamming distance was the approach chosen to match the descriptors. Therefore, the bit difference between two descriptors measures their disparity.

2.2.7 ORB - Oriented FAST and Rotated BRIEF

ORB is an algorithm for feature detection and description and uses FAST and BRIEF as keypoint detector and descriptor, respectively [42]. In this approach, it is presented the FAST method with an orientation component added that is obtained using a corner orientation method called *intensity centroid* which, for instance defines that the intensity of a corner is offset from its center and the resulting vector may be used for determining an orientation [42]. Moreover, it is also presented a rotation-aware version of BRIEF for increasing the performance of this method with in-plane rotation. This version steers BRIEF according to the orientation of keypoints, so as long as the keypoint orientation is consistent, the correct set of "steered" points is used to construct the descriptor. These two adapted methods together form a keypoint detector and descriptor where rotational invariance is achieved, but not scale invariance.

2.2.8 FREAK - Fast Retina Keypoint

FREAK is keypoint descriptor and is inspired on the human visual system and somewhat mimics the human retina [43]. Similarly to BRISK, FREAK makes use of a circular area for comparing pairs of pixel intensities, but this one differs from the other by having a higher density of points at the center region. In this method, a performance improvement was achieved by using an exponential variation of the Gaussian kernels size and by overlapping the receptive fields. The descriptor, such as for BRIEF and BRISK, is binary and is formed by a string of one-bit DoGs. To select discriminant pairs that effectively describe an image, an algorithm similar to ORB is performed. The selected pairs appeared to be structured towards a coarse-to-fine ordering: the first set of pairs represents the peripheral receptive fields and the coarse information; and the other one represents more centered receptive fields and fine information. This is identical to the behaviour of the human eye since firstly it uses the perifoveal area, which contains less densely distributed receptive fields therefore it captures less detailed information, to estimate the location of an object of interest. Then, the foveal area, which contains more densely distributed receptive fields thus

capturing more detailed information, is used to validate the previous step. The descriptor matching is accomplished by doing a saccadic search such as done by humans. This procedure starts by searching and comparing the first 16 bytes of the descriptor with a threshold. If the distance is smaller, the comparison continues with the next bytes to analyse higher resolution information, resulting on a sequence of comparisons that speeds up the matching. The estimation of the keypoint orientation is performed summing the local gradients of the selected pairs being this analogous to BRISK [43], since the latter makes use of long pairs to obtain the orientation while in FREAK the pairs selection is based on the receptive fields symmetry with respect to the center.

2.2.9 KAZE

KAZE is a method for feature detection and description in nonlinear scale spaces [44]. This method instead of using Gaussian blurring as SIFT, which does not respect the boundaries of objects, uses a nonlinear diffusion filtering combined with the conductivity function proposed by Perona and Malik [45], along with additive operator splitting techniques in order to keep the edges unaffected. This process is computationally costly due to the fact that the aforementioned filtering requires solving a set of differential equations that are solved using the already mentioned numerical method called additive operator splitting. In respect to keypoint detection, the algorithm computes the response of the determinant of Hessian at multiple scale levels. The response of the detector is analysed at different scale levels, so the highest responses in scale and spatial location are possible keypoints. For finding the dominant orientation and therefore obtaining rotation invariant descriptors, it is necessary to estimate the dominant orientation in a local neighbourhood whose center is at the keypoint location [44]. Having the dominant orientation, the keypoint descriptor vector is built according to it (doing the computation of the derivative responses centered at the keypoint) and is normalised into a unit vector becoming contrast invariant, accomplishing scale and rotational invariance.

2.2.10 AKAZE - Accelerated KAZE

AKAZE is an accelerated version of KAZE that speeds up the computation process of the nonlinear scale space using Fast Explicit Diffusion [46]. Moreover, this algorithm utilises binary descriptors using a modified version of Local Difference Binary descriptor [47], that prospects intensity and gradient information from the nonlinear scale space.

2.2.11 HOG - Histogram of Oriented Gradients

HOG is a feature descriptor and is based on the evaluation of normalised local histograms of oriented gradients of images [48]. The method starts with the input image being divided into small spatial areas (cells), and for each cell the oriented gradients are calculated over its pixels. Every pixel has a gradient with magnitude and direction components. Then, an histogram of gradients is created in each cell as a representation of each of them. To obtain invariance for changes in

light conditions, a normalisation is done by aggregating the local histogram "energy" measure over larger spatial areas (blocks) and using the results to normalise all cells in the block [48].

2.2.12 Comparative Analysis of the Algorithms

In order to choose the best algorithm of those previously presented, a careful analysis must be made, so in Table 2.2 is shown an overview of the algorithms concerning whether they own (Y - Yes) or not (N - No) certain characteristics. Moreover, in [49] is concluded that SIFT, SURF and BRISK are the most invariant to scale and ORB is the least. SIFT, BRISK, KAZE and AKAZE are more accurate with respect to image rotations. ORB and BRISK, when compared to the others, present the better invariance to affine changes and beyond that, are the most efficient algorithms. In general, SIFT and BRISK appear to be the algorithms with highest accuracy regarding all types of geometric transformations, with SIFT being the most accurate one [49]. It is important to notice that the absence of FREAK and HOG above is due to these being just and only feature descriptors, however both descriptors could potentially outperform some descriptors of the other algorithms.

Table 2.2: Summary of the feature detection and description algorithms.

Method	**Feature Detector**	**Feature Descriptor**	**Scale Invariant**	**Rotation Invariant**
SIFT	Y	Y	Y	Y
SURF	Y	Y	Y	Y
FAST	Y	N	Y	N
AGAST	Y	N	Y	N
BRIEF	N	Y	N	N
BRISK	Y	Y	Y	Y
ORB	Y	Y	N	Y
FREAK	N	Y	Y	Y
KAZE	Y	Y	Y	Y
AKAZE	Y	Y	Y	Y
HOG	N	Y	Y	N

2.3 Machine Learning

Machine learning is an Artificial Intelligence domain that focus on teaching a machine to learn from input data [50]. The way data are handled by the machine and their purpose represent the types of learning that define machine learning methods and can be distinguished into four main categories:

1. Supervised learning: this type of machine learning algorithms rely on training a model from labelled input data to obtain a function that is capable of producing a reasonable output when it is fed with "unseen" data. The input dataset is divided into train and test dataset. During the training, the algorithm learn some patterns from the training dataset and then, the resulting model is evaluated with the test dataset, where it will predict and classify new

data [50]. If the behaviour of the model is worse than what is aimed, some actions can be made before training again: parameter tuning, definition of a new dataset, identification of new relevant data or selection of a new algorithm [50]. On the other hand, if the performance of the model is similar or even better than the expected, a new classifier is produced.

2. Unsupervised learning: algorithms of this type learn features from unlabelled data as a way of structuring them. Instead of generating an output, these algorithms use the learned features to group parts of the input data according to their commonalities. The main uses of this type of learning are clustering and the reduction of features [50].

3. Semi-supervised learning: this type is a combination of supervised and unsupervised learning. It can be helpful in the cases where is necessary to obtain labels for not categorised data. Beside that, improvements in the learning task can occur when unlabelled and labelled data are placed together [50].

4. Reinforcement learning: this type of learning regards the way that an agent makes decisions, based on the actions to take in the environment, in order that the outcome is the most rewarding. When the agent is put into a new situation, it has no knowledge about the actions to take and, their consequences may affect the environment and its future actions [50]. This method is based on trial and error search and delayed outcome.

2.3.1 SVM - Support Vector Machine

SVM is a supervised machine learning method originally designed to solve classification problems with two classes [51]. The idea behind this method is the mapping of input vectors, through a linear function, into a high dimensional feature space and, in this space, a linear decision surface is built to enhance the generalisation among the classes.

Regarding multi-class classification using SVM, there are some implementations based on several binary classifiers: "one-against-all", "one-against-one" and Directed Acyclic Graph (DAG) SVM.

One-against-all is a method that constructs k SVM models corresponding to k classes. The SVM model m is trained with the examples in the class m with positive labels, and the other examples with negative labels [52]. Therefore, considering l training data $(x_1,y_1),...,(x_l,y_l)$, where $x_i \in R^n$, $i = 1,...,l$ and $y_i \in \{1,...,k\}$ is the class of x_i, the model m solves the problem defined by (2.11), where the function ϕ maps the training data x_i into a high dimensional space and C is the

cost parameter.

$$\begin{aligned} \min_{w^m,b^m,\xi^m} \quad & \frac{1}{2}(w^m)^T w^m + C\sum_{i=1}^{l}\xi_i^m \\ & (w^m)^T\phi(x_i) + b^m \geq 1-\xi_i^m, \text{ if } y_i = m, \\ & (w^m)^T\phi(x_i) + b^m \leq -1+\xi_i^m, \text{ if } y_i \neq m, \\ & \xi_i^m \geq 0,\ i = 1,...,l \end{aligned} \tag{2.11}$$

The minimisation of $\frac{1}{2}(w^m)^T w^m$ is equal to the maximisation of $2/\|w^m\|$ which corresponds to the margin of two classes. The cost term $C\sum_{i=1}^{l}\xi_i^m$ can reduce the number of training errors when the data are not linear separable [52]. The search for a balance between the two terms previously mentioned is the concept that SVM is based on.

After solving (2.11), k decision functions appear in the following fashion:

$$(w^1)^T\phi(x) + b^1\ ,...\ (w^k)^T\phi(x) + b^k$$

The class which x belongs to, is the one that has the biggest of the decision function [52], as is shown in (2.12).

$$\text{class of } x \equiv \text{argmax}_{m=1,...,k}((w^m)^T\phi(x) + b^m) \tag{2.12}$$

Another method named one-against-one builds $k(k-1)/2$ classifiers, being that each one is trained with two classes derived data [52]. The binary classification problem in (2.13) must be solved for the training data of classes i and j.

$$\begin{aligned} \min_{w^{ij},b^{ij},\xi^{ij}} \quad & \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_{t}\xi_t^{ij} \\ & (w^{ij})^T\phi(x_t) + b^{ij} \geq 1-\xi_t^{ij}, \text{ if } y_t = i, \\ & (w^{ij})^T\phi(x_t) + b^{ij} \leq -1+\xi_t^{ij}, \text{ if } y_t = j, \\ & \xi_t^{ij} \geq 0 \end{aligned} \tag{2.13}$$

For the classes attribution, a voting strategy called "Max Wins" can be used. This technique is based on the following procedure: if $((w^{ij})^T\phi(x) + b^{ij})$ is signed, x belongs to class i and the vote counter of this class is increased by one. Otherwise, class j is the one to which will be added one vote. Then, x will belong to the class that gather more votes [52].

The last method is called Directed Acyclic Graph Support Vector Machines (DAGSVM) and its training phase is analogous to the one-against-one method, thus resolving $k(k-1)/2$ binary SVMs. As opposed to that method, the testing phase of DAGSVM makes use of a rooted binary directed acyclic graph that has $k(k-1)/2$ internal nodes and k leaves [52]. Each node represents

a binary SVM of classes i and j, and being x a test sample, the operation starts at the root node where the decision function is evaluated. Then, depending on the output, it moves either left or right, leading to a path that when reaches a leaf, allows determining the class of x [52].

The results obtained in [52], allow to conclude that the "one-against-one" method is the most competitive. The latter and DAG are considered the more appropriate methods for practical utilisation [52].

2.3.2 Deep Learning

Deep learning is an area of machine learning based on artificial neural networks. The methods of deep learning are representation-learning methods, that in turn are composed by a set of methods which grants the discovery of representations necessary for detection or classification by a machine just by feeding it with raw data [53]. These methods along with multiple levels of representation, obtained from the constitution of modules that transform a basic representation (raw data) at a lower level into a higher-level representation, form the deep learning methods.

Regarding detection tasks, the higher layers emphasise the most import aspects from the input data to accurately identify and disregard existing variations [53]. For instance, after an image being processed, in the lower layers the resulting learned features comprise the edges, while higher layers may unveil more relevant information to a human such as objects.

2.3.2.1 YOLO - You Only Look Once

YOLO is an object detection system for real-time processing that makes use of a neural network to identify objects in images [54]. This system starts by resizing the input image to 448×448, then the image is divided into a $S \times S$ grid, being that each grid cell will predict only one object. The system, for each grid cell and through a single neural network, predicts B bounding boxes, each one having a confidence score, and C class probabilities for each. The confidence score represents the probability the box containing an object and the accuracy of the boundary box [54]. The class probability is the probability of a detected object belonging to a class. This procedure is illustrated in Figure 2.24 and its predictions are encoded as a tensor defined by $S \times S \times (B \times 5 + C)$.

Lastly, to fix the problem of YOLO generating duplicates for the same object, a non-maximal suppression is applied. Some of the benefits of YOLO over other object detection methods are: it is fast, running at 45 frames per second; it is good for real-time processing; its predictions of object location and classes are made using one single network; and it is capable of learning any type of object, being robust when applied to unexpected inputs.

In 2017, a second version of YOLO, *YOLOv2*, appeared with improvements regarding detection accuracy and speed. This version is also called *YOLO9000* due to be capable of predicting more than 9000 different object categories and yet performing in real-time [55].

The current version of YOLO is *YOLOv3* [56] which presents an even better performance than the previous versions with respect to speed.

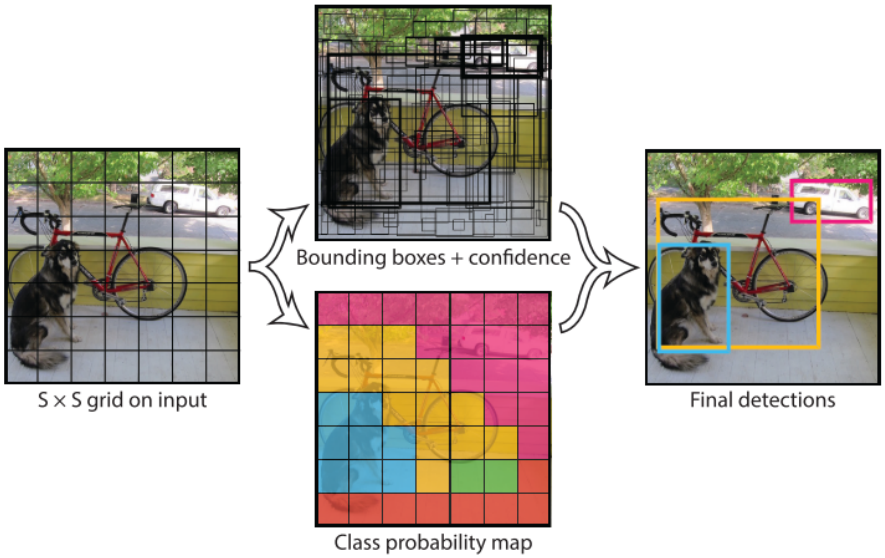


Figure 2.24: The model that YOLO is based on [54].

2.4 Evaluation Metrics

The evaluation of results obtained from a model is normally performed using evaluation metrics such as Precision, Recall and F-measure. Before defining such metrics it is worth to clarify the meaning of some names used in this domain. Therefore, True Positives (TP) and False Positives (FP) specify the correct and incorrect predicted positives, respectively. Similarly, True Negatives (TN) and False Negatives (FN) refer to the correct and incorrect predicted negatives. Regarding the metrics, **Precision** (p) is the percentage of predicted positive cases that are effectively real positives and **Recall** (r) is the percentage of real positive cases that are effectively predicted positives [57]. These metrics are defined in (2.14) and (2.15).

$$\text{Precision}: \quad p = \frac{TP}{TP+FP} \tag{2.14}$$

$$\text{Recall}: \quad r = \frac{TP}{TP+FN} \tag{2.15}$$

Nevertheless, these measures have a problem that is when enhancing one of them, the other gets worse, so a trade-off must be made to balance both. To tackle this is used an aforementioned metric called **F-measure** (F). This metric considers both Precision and Recall thus one can aim to maximise this measure in order to improve a model instead of balancing two separate measures [57]. This metric is defined by the harmonic mean of Precision and Recall [57] as is shown in (2.16).

$$\mathrm{F} = 2 \times \frac{p \times r}{p+r} \tag{2.16}$$

2.5 Development Technologies

This section introduces some of the technologies that were used during the development of this dissertation for building the tiles and programming the image processing part.

Concerning the tiles, *OpenSCAD* [58] and *Ultimaker Cura* [59] are both software for 3D printing. The first is used to create 3D CAD models while the second aims at the slicing and preparation of the models for printing.

Regarding the image processing part, *OpenCV* is the computer vision library chosen for being an open-source library and having several computer vision algorithms [33]. This library is written in C++, but it also have some of its modules available for other languages such as Java, Python and JavaScript. Furthermore, JavaScript was the preferred programming language due to the easier integration with the Tactode system. JavaScript (JS) is a *just-in-time* compiled programming language, meaning that the code is compiled during the execution of the program, and is a single-threaded, multi-paradigm and prototype-based language as well [60]. In addition, *Node.js*, which is an asynchronous event-driven JS run-time [61], is selected because of its ability to use JS on a non-browser environment. Besides that, Node.js provides several open-source packages through npm (Node Package Manager) [62], being that there is one in particular, *opencv4nodejs* [63], that is considered useful because, currently *OpenCV* does not have bindings for all of its functionalities in JS, so as an alternative this package was also taken into account.

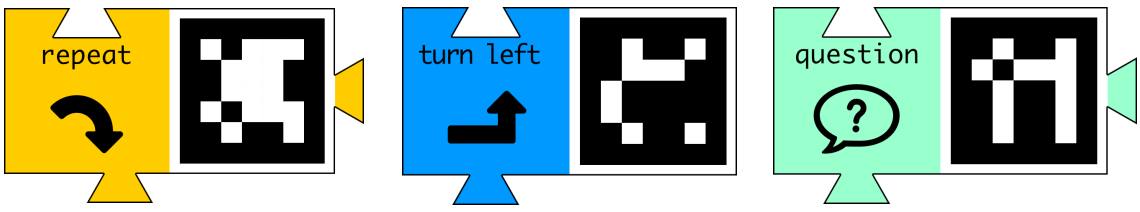
Chapter 3

Problem Statement and Solution Proposal

This chapter describes the problem inherent to this dissertation and the solution implemented to solve it. This solution is divided in two main parts: one concerning the Tactode tiles and another one related to the software algorithm. Also is presented some experiments made regarding integration purposes.

3.1 Problem Definition

Currently, the tile recognition phase of Tactode Programming System is based on the detection and processing of ArUco markers. Figure 3.1 shows some of original Tactode tiles and one can observe that each tile is composed by an ArUco marker. Each marker of this type has an embedded code resulting, as mentioned in Section 2.1.4, from the barcode-based design of these markers, allowing tiles to be identified through it.

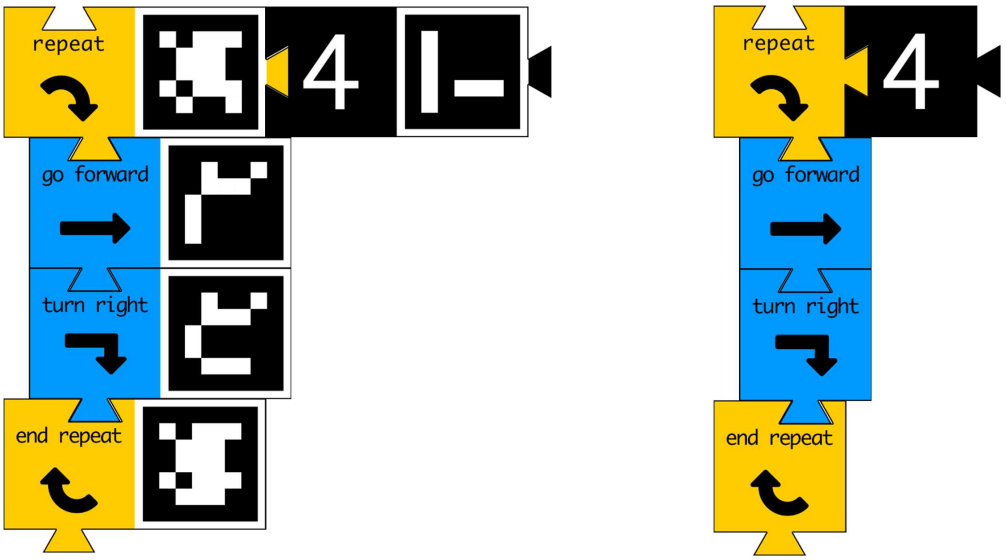


(a) (b) (c)

Figure 3.1: Examples of original Tactode tiles: (a), (b) and (c) are the "repeat", "turn left" and "question" original tiles, respectively.

Thus, in this dissertation, the problem at hand consisted of detecting and recognising the Tactode tiles present in a code by their own content instead of using the ArUco markers. To achieve this, the tiles had to be redesigned to remove the markers. To have a visual perception of this decision effects, in Figure 3.2 is shown two different approaches - tiles with and without ArUco

markers - that represent the same Tactode code example. As can be verified, there were improvements in terms of occupied area, since the area of each tile decreases almost to half with the removal of the markers.



(a) (b)

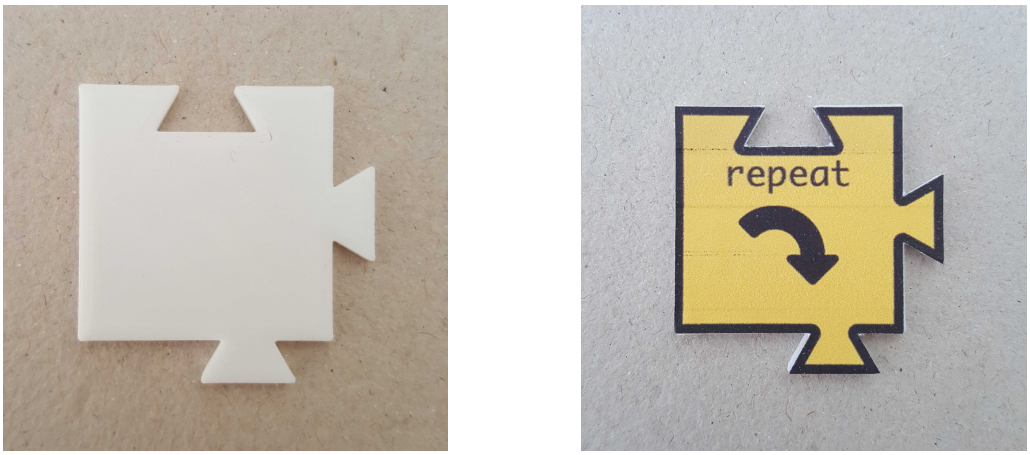
Figure 3.2: Two examples of Tactode codes for taking a square-shaped route: (a) and (b) represent the same code with and without ArUco markers, respectively.

3.2 Proposed Solution

In this section is presented the proposed solution to tackle the problem already introduced in Section 3.1, which is divided in two parts: Hardware, that corresponds to building new tiles without ArUco markers and Software, that is the developing of a vision-based algorithm capable of detecting and identifying the new tiles.

3.2.1 Hardware

Concerning the original Tactode tiles, they are made of Ethylene Vinyl Acetate (EVA) having 2 mm of thickness [1]. Similarly, the new tiles have the same thickness, but they were 3D printed using Polylactic Acid (PLA) and then, coloured printed paper was glued on top, as can be seen in Figure 3.3.



(a) (b)

Figure 3.3: The tile "repeat" without and with coloured printed paper on top: (a) is the tile resulting from 3D printing and (b) is the same tile after being glued with coloured printed paper.

The dimensions of the new tiles also differ from the original ones, being the first smaller in terms of occupied area. From Figure 3.4, is possible to conclude that the occupied area (no "teeth" included) of the new tiles are 28% smaller then the original tiles.

In addition, as can be observed in Figure 3.4 at right, was added a 1 mm border to each tile (white border for the black tiles and black border for all the other) in order to facilitate the segmentation step of the vision algorithm. This way, due to the existent contrast between the colour of the tile and the border, the problem of two tiles with the same colour being together and possibly being wrongly processed as one individual tile is mitigated.

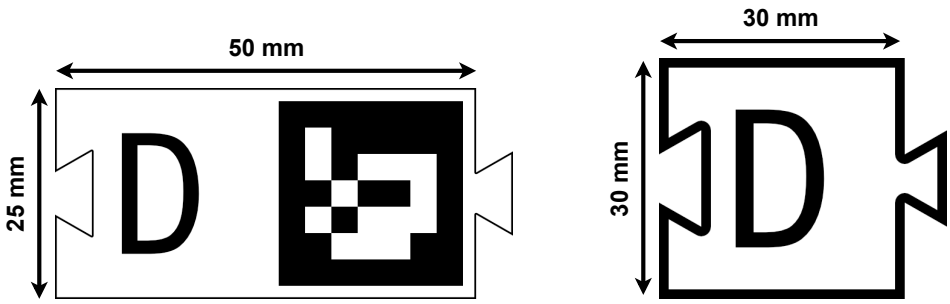


Figure 3.4: Comparison of dimensions between original tiles and new tiles.

Furthermore, to avoid classification problems of tiles, some tiles were changed because of their similarity with others, namely "forever" and "end forever" tiles were changed since these were very similar to "repeat" and "end repeat" tiles, respectively, as it shows Figure 3.5. The main changes on these tiles were with respect to the symbol of them and such modifications can be seen in Figure 3.6. Finally, were built 34 new tiles (32 of which are different), that are presented in Figure 3.7, to test and validate the software part which will be described next.

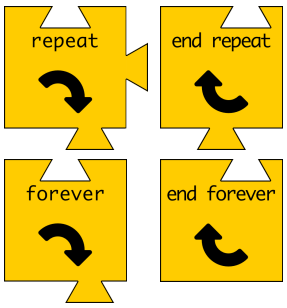


Figure 3.5: Similitude between the "repeat" and "end repeat" tiles with the "forever" and "end forever" tiles, respectively.

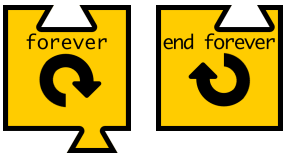


Figure 3.6: Modifications done on "forever" and "end forever" tiles.

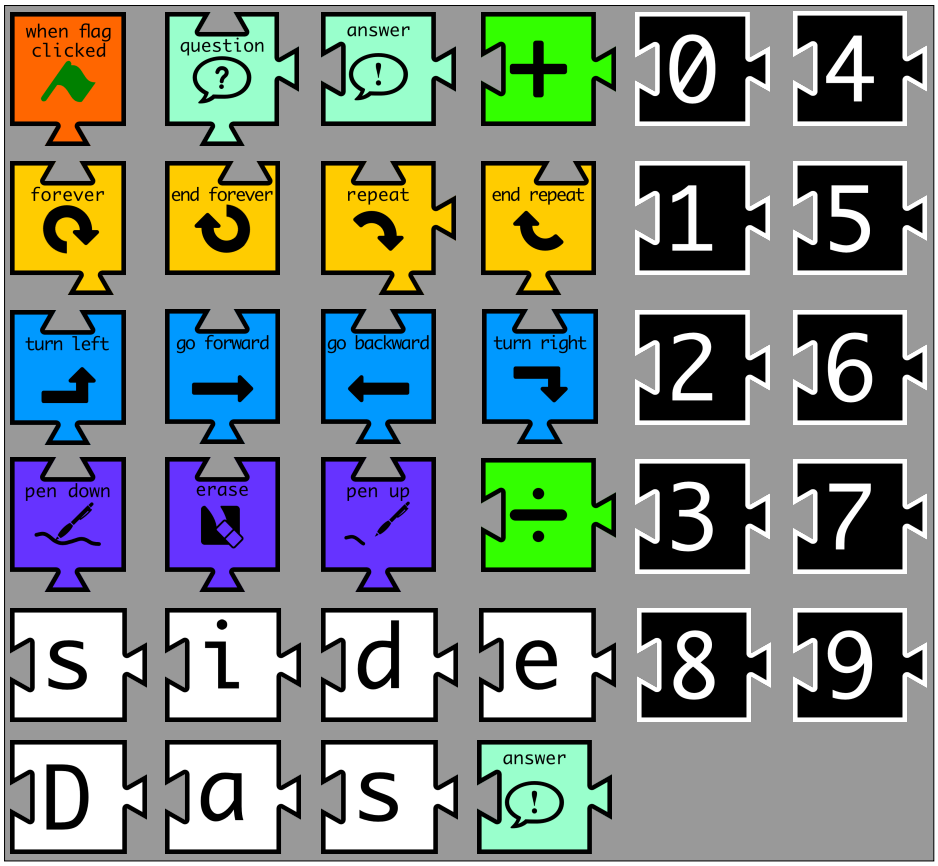


Figure 3.7: New tiles that were built.

3.2.2 Software Pipeline

In this section, the pipeline that composes the computer vision algorithm is described. The steps that compose this pipeline are shown in Figure 3.8. The algorithm starts by receiving an input image of a photographed Tactode code that goes through some middle stages and ends up with a matrix whose elements are classified tiles which are positioned in the matrix accordingly with the pictured code. Every stage of this process will be presented in the following sections. One thing that must be said is that all the stages that will be described were developed and are based on the node package *opencv4nodejs* already mentioned in Section 2.5.

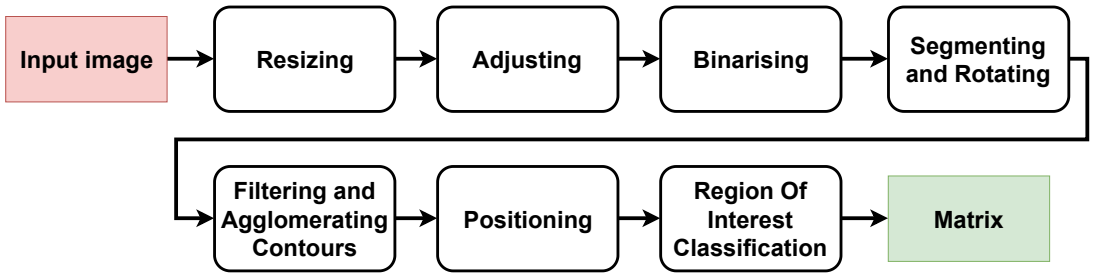


Figure 3.8: Block diagram of the software pipeline.

3.2.2.1 Resizing

After receiving and reading the input image, it is resized to a normalised resolution. First, the bigger side of the image is found and is assigned to it 1920 px length. The smaller side follows the previous length respecting the aspect ratio of the original input image.

Therefore, considering H_O as the input image original height and W_O as its width, the aspect ratio will be equal to $\frac{W_O}{H_O}$ if $H_O \geq W_O$. In this case, the new height, H_N, will be set to 1920 px and the new width, W_N, will be equal to $H_N \times \frac{W_O}{H_O}$. In the opposite case, where $H_O < W_O$, the result would be, $W_N = 1920$ px and $H_N = W_N \times \frac{H_O}{W_O}$. These operations are calculated only if both the width and height lengths of the input image are different than 1920 px, being that the image is not resized in this case. Also the resolution of 1920x1080 px was considered to be the smallest that this algorithm would have to deal with, so any resizing that has to be done will always be down-sampling. Consequently, the interpolation method chosen to perform this operation was Area interpolation since it is the best for image decimation [1] [33]. In Figure 3.9 is presented the resized code that will be used in the next stages.

[1] OpenCV. Geometric Image Transformations. Available at https://docs.opencv.org/4.4.0/da/d54/group__imgproc__transform.html. Last accessed 2020-07-05.

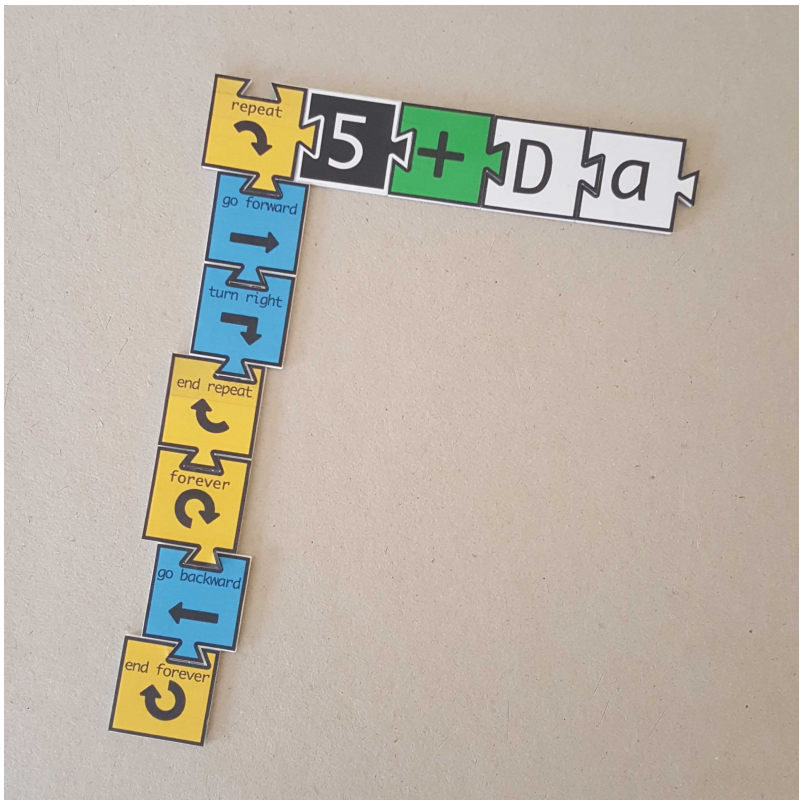


Figure 3.9: Resized Tactode code.

3.2.2.2 Adjusting

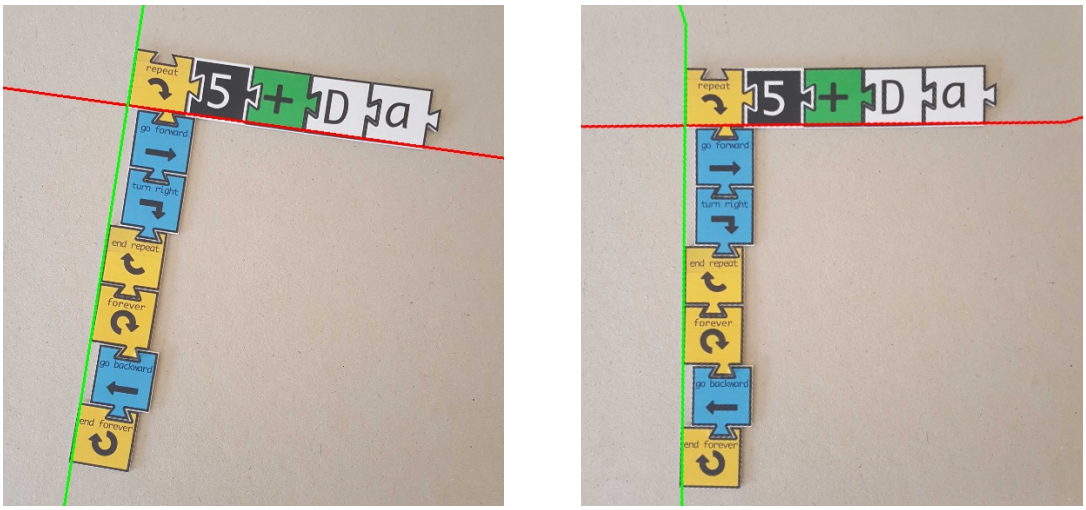
Since any photographed code will never be perfectly oriented and will exhibit, most of the times, at least some minimal rotation, in this stage the resized image previously computed is adjusted to the closest of the rotation angles 0º, 90º, 180º and 270º. The threshold that defines if the initial rotation angle is close enough to those angles is the angle that divides evenly the pair of angles among which the initial angle is, e.g., if a code have an initial rotation angle, R_i, equal to 100º, the pair of angles that it will be between are 90º and 180º, and the respective threshold, T, will be 135º, therefore the code will be adjusted to 90º (the image will be adjusted by -10º), as $R_i > T$ thus proving the proximity of R_i to this angle.

For this purpose, the main method used was Hough lines. The procedure starts by making a copy of the resized image, followed by a smoothing using a Gaussian filter with a 9x9 kernel, a down-sampling using Area interpolation to 30% of the height and width (resulting nearly in a 576x324 image) and then, is performed a conversion to grey scale and is applied a Canny operator to evidence the edges of the code. From the previous operations a binary (black and white) image is obtained on which Hough lines will be calculated.

After having the lines, they are filtered to get the two biggest that are approximately perpendicular to each other. In case these two be found, the adjusting angle will be the mean value between the angles of the lines. If none perpendicular line is found, then the adjusting angle will only be

the value of the biggest line. Then, the resulting adjusting angle is used to rotate the resized image received at the beginning of this stage.

This strategy based on two lines was chosen to improve the adjustment accuracy particularly in cases where the photos have perspective and these two lines are "less" perpendicular than in frontal photos. In Figure 3.10, one can see the code before (with the detected Hough lines) and after the adjustment being made.



(a) (b)

Figure 3.10: Two different images of the same Tactode code: (a) is the code before the adjustment and (b) is the code after.

3.2.2.3 Binarising

The binarisation of the adjusted image is accomplished by morphological operations. These are image shape-based operations which rely on a structuring element or kernel [2] [33]. First, the adjusted image is smoothed by a 11x11 Gaussian kernel and then, using the structuring element Cross with size 6x6, is performed a Gradient morphological operation to evidence the edges of the code. At this point, a colour thresholding is made originating a binary (black and white) image.

However, the image can have some noise and potentially some holes in the foreground objects, thus morphological operations of Opening and Closing are executed to solve, respectively, both issues [2] [33]. Figure 3.11 shows the resulting image of this stage.

[2] OpenCV. Morphological Transformations. Available at https://docs.opencv.org/4.4.0/d4/d76/tutorial_js_morphological_ops.html. Last accessed 2020-07-05.

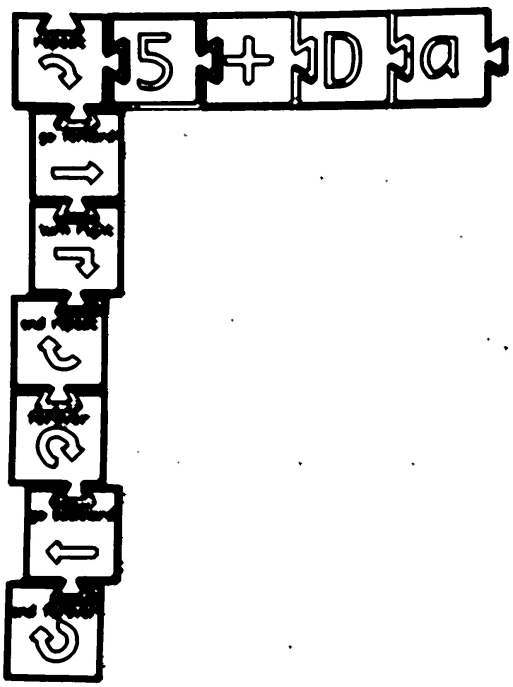


Figure 3.11: Binary image containing the Tactode code.

3.2.2.4 Segmenting and Rotating

The binary image obtained from the preceding step can arrive, at this stage, rotated 90, 180 or 270 degrees clockwise. So, in order to the following steps of the pipeline work correctly, the image has to be rotated to the 0º corresponding position.

Before tackle this, some segmentation operations are executed to split the image in parts that can be further used for detection and classification purposes. The segmentation method chosen was contours because its implementation in *OpenCV* is supported by various useful properties and methods, already shown in Section 2.1.5, and also its easiness of use.

Initially, the contours are found using the retrieval mode *TREE*, returning all contours and their hierarchies. Then, they are filtered to find the "grandparent" contour and its bigger (in terms of area) "child", the "parent" contour, whose children are the most interesting for the problem at hand. The grandparent contour represents the main frame where the Tactode code was photographed, the parent contour surrounds the code, thus including all constituent tiles, and its inner contours are the children which are directly related to the tiles. Figure 3.12, shows those three types of contours distinguished by colours.

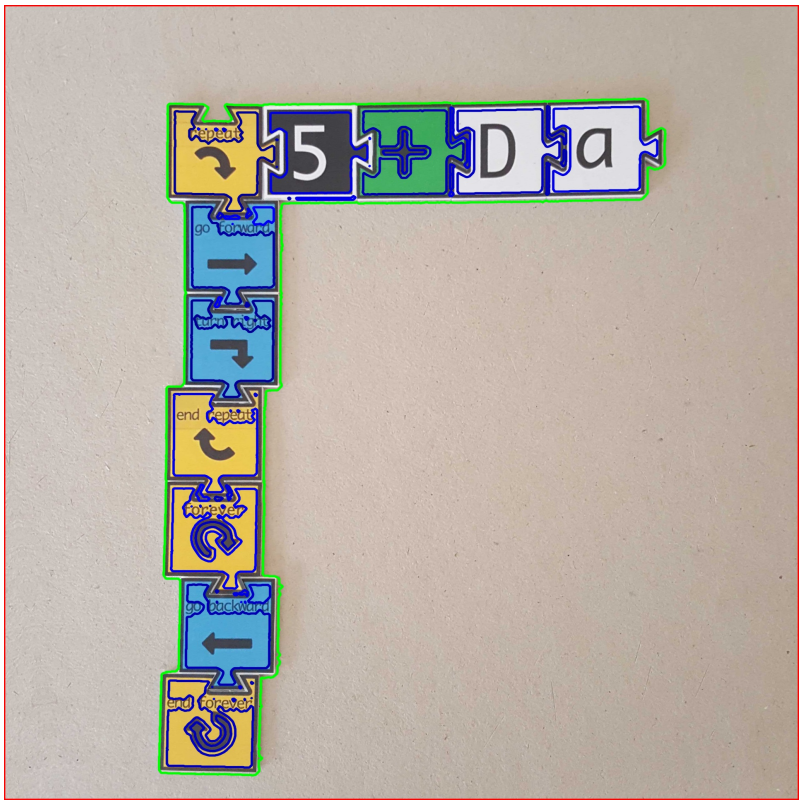
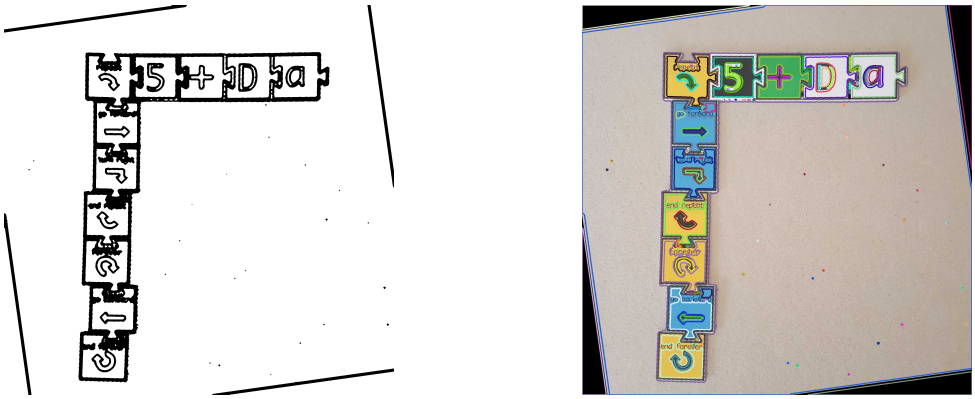


Figure 3.12: The main contours: the grandparent (red), parent (green) and its child contours (blue).

Sometimes, in addition to the main frame (where the code resides) some parts external to it appear, as illustrated in Figure 3.13. This happens mainly after the adjustment (explained in Section 3.2.2.2) is done when using the pre-built *OpenCV* library for JS on the web, instead of the Node package *opencv4nodejs* on Node.js. Those parts belong to the same hierarchy level of the main frame and must be filtered. This filtering is performed using the hierarchy array of the contours that was already mentioned and explained in Section 2.1.5. So, the third value of the hierarchy array (*First_Child*) was used, since those parts do not have any tile or code inside them, hence do not have any contour and, consequently, this value are equal to -1. Also, the fourth value (*Parent*) was used to ensure about which contours were the outermost meaning that they (the main frame and the possibly existent parts) do not have parent, thus this value must be equal to -1 in these cases. These two values along with the contours being sorted by their area in a descending way, helped to obtain the grandparent, parent and child contours previously referenced in this section. In Algorithm 1, is shown this procedure where the set of contours are arranged in an array and then, filtered according to what was mentioned above.



(a) (b)

Figure 3.13: Two images of the same code after the adjustment with extra contours: (a) is the binary image and (b) is the image with all found contours. At (b), in black, are the external parts that belong to the same hierarchy level of the main frame.

Algorithm 1: Contours reducing

```
Result: childArr[] containing the child contours
set childArr[];
sort contoursArr[] by descending area order;
set grandparent_idx and parent_idx to -1;
for each contour in contoursArr[] do
    get hierarchy[] and idx from contour;
    if grandparent_idx == -1 && hierarchy[4] == -1 && hierarchy[3] != -1 then
        set grandparent_idx to idx;
    else
        if parent_idx == -1 && hierarchy[4] == grandparent_idx then
            set parent_idx to idx;
        else
            if hierarchy[4] == parent_idx then
                push contour to childArr[];
            end
        end
    end
end
```

After reducing all contours down to the interesting ones (the children), another filtering operation needed to be done before starting the phase where the rotation issue, previously mentioned in this section, will be attended. This last filtering boils down to identifying which child contours approximately represent tiles, i.e., whether they are big enough (in terms of area) to be considered as a tile or, on the contrary, noise. Therefore, the largest child contour is found and then, all the other are compared to it with a 35% area threshold and filtered if they fell under the threshold, resulting on a set of tile alike contours, as can be seen in Figure 3.14.

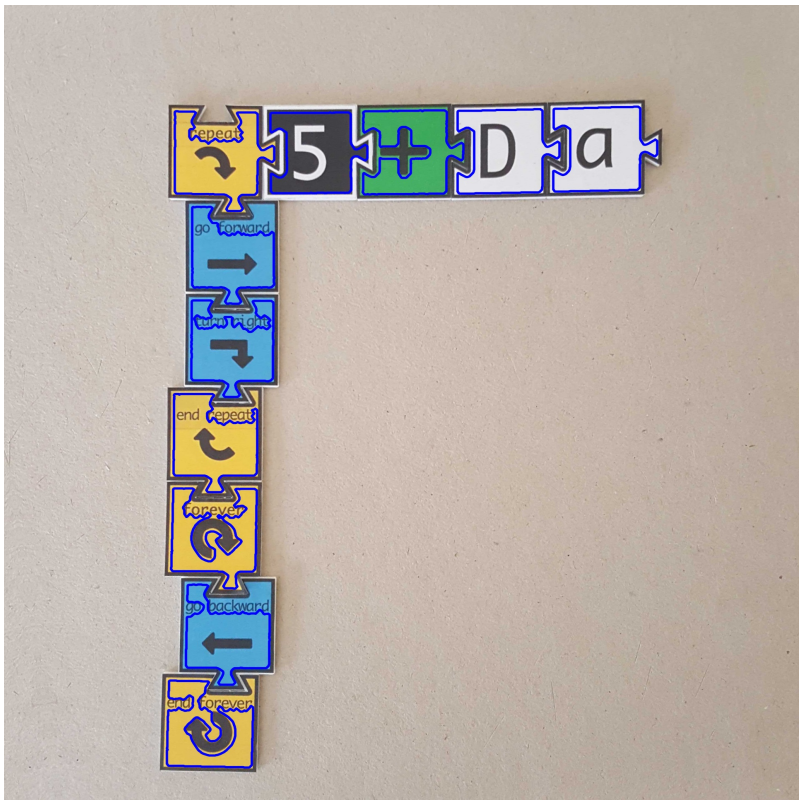


Figure 3.14: The most similar contours to the tiles.

Regarding the question of the image being rotated, two different approaches emerged. The first approach starts by computing each contour moments to extract their centroids which are calculated through (3.1), where M_{00}, M_{10} and M_{01} are spatial moments.

$$C = (x_C, y_C) = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right) \tag{3.1}$$

Then, since every tile has at least one "tooth", by comparing the contour centroid with the center of the contour bounding box, it should result in a slight deviation of one another because the tooth pull the center of the bounding box towards its direction more than the centroid. To a better understanding, Figure 3.15 illustrates this operation.

At this point, to know the direction of some tile tooth, some conditions, specified in Table 3.1, had to be followed.

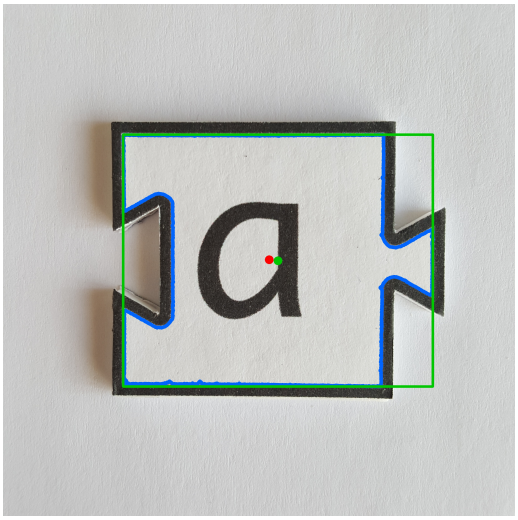


Figure 3.15: Positions of the contour centroid and its bounding box center in the tile "lower a": in blue the contour, in red its centroid and, in green its bounding box and the respective center.

Table 3.1: Conditions for determining the direction of a tooth of a tile.

Direction	Condition
Right	$x_{centroid} < x_{center}$, $y_{centroid} \approx y_{center}$
Down	$x_{centroid} \approx x_{center}$, $y_{centroid} < y_{center}$
Left	$x_{centroid} > x_{center}$, $y_{centroid} \approx Y_{center}$
Up	$x_{centroid} \approx x_{center}$, $y_{centroid} > y_{center}$

With the capability of deducing the tooth direction of any tile, the rotation of a complete Tactode code must also be determined. In Table 3.2, are presented, for each rotation angle, the possible directions that the teeth of the tiles composing the code can present, allowing to determine whether the code is rotated and if so, what is its angle of rotation.

Table 3.2: Possible teeth directions for determining the rotation angle of a code.

Rotation (clockwise)	Possible directions
0º	Down and Right
90º	Down and Left
180º	Up and Left
270º	Up and Right

Therefore, having the centroids and bounding boxes centers computed, the direction are determined for each contour and, according to the rules presented in Table 3.2, the two most voted directions would define the rotation angle of the code.

Nevertheless, this approach did not work because most of the times the contours did not cover the whole tile, hence affecting the centroid location, resulting in the directions being wrongly determined and also the rotation angle.

The other approach chosen for this problem, that in turn worked, consists of detecting teeth pointing down in the Tactode tiles and the code is rotated until reaching the 0º position which is determined by two rules.

It can be said that every non-rotated Tactode code is composed by a main column of tiles and, eventually, some rows. The Tactode codes rotated by 0º and 90º clockwise are the only ones possibly having tiles with teeth pointing down. The first, in the case of having rows, their tiles will always be to the right of the main column. On the other hand, if the second one have tiles with teeth pointing down, that means that it has a least one column (rotated row) and the other tiles (outside the column) will always be at the left of that column. Therefore, the two rules that determine whether the code is at the 0º position are specified as the following:

1. there must be at least two tiles with the teeth pointing down and forming a column;
2. all the other tiles must be in the column or to its right.

To find teeth pointing down, we divided vertically each contour bounding box by three and looked for a tooth at the lower part. Figure 3.16 shows the ROI - region where a tooth can be found - in two different tiles. The teeth detection is based on a combination of image descriptors with machine learning, namely HOG and SVM. This is a well-known method for object image detection and it was used on [48] for pedestrian detection.

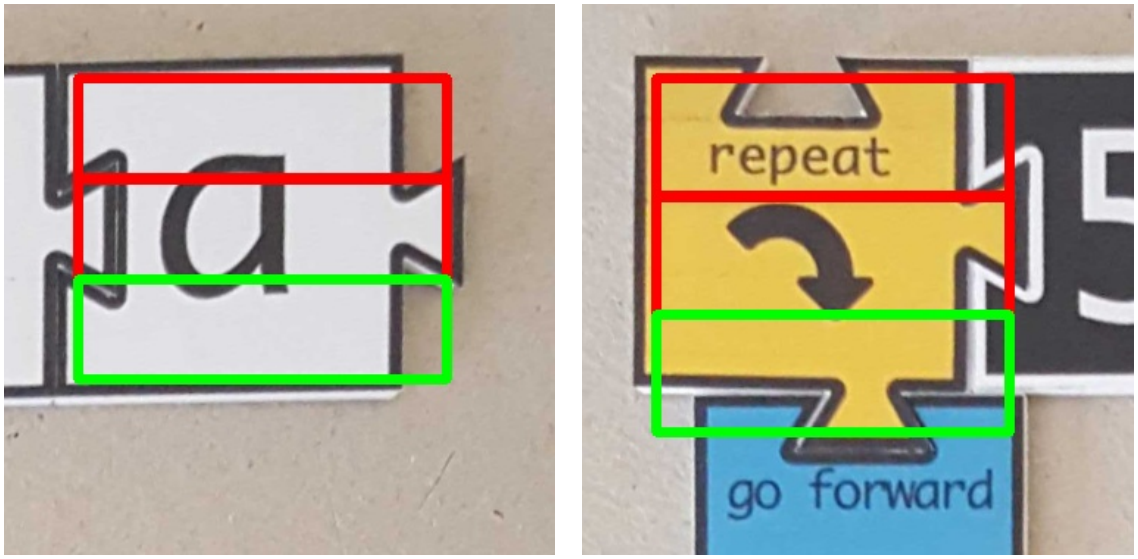


Figure 3.16: Two images of extracted tiles from the code with coloured regions: in green is the most interesting region to find a tooth.

A dataset of 2100 positive samples (with teeth) and 2430 negative samples (with no teeth) was built and then a linear SVM (C = 0.1) was trained using the HOG descriptors vectors calculated over each dataset sample. The HOG descriptor configuration follows a window size of 32x64 px that is divided in cells of 8x8 px and in blocks of by 16x16 px (or 2x2 cells). The block stride corresponds to 8x8 px (or one cell), the gradients are calculated with sign and each histogram is divided in nine bins, resulting in a descriptors vector length of 756 per sample.

The training consisted of extracting the ROI from each image sample, adjusting the height to half the size of the width (in order to give a 1:2 ratio defined by the 32x64 resolution of HOG window) then, the ROI is resized to 32x64 px followed by the SVM being fed with the computed descriptors vector. The procedure that goes from reading the image of the tile to obtaining the ROI with a 1:2 ratio is shown in Figure 3.17. After the training, a 10-fold cross validation was performed resulting in zero errors.



Figure 3.17: Three images of the tile "flag" representing three different steps of ROI extraction: on the left side the photograph of the tile, in the middle the region of the tile and, on the right side the ROI.

With the teeth detector trained, two generated files containing the configuration of HOG and the trained model are saved to be loaded and used later to detect teeth. The procedure to find whether the code is rotated and, if so rotating it until the 0º position is reached, is presented below:

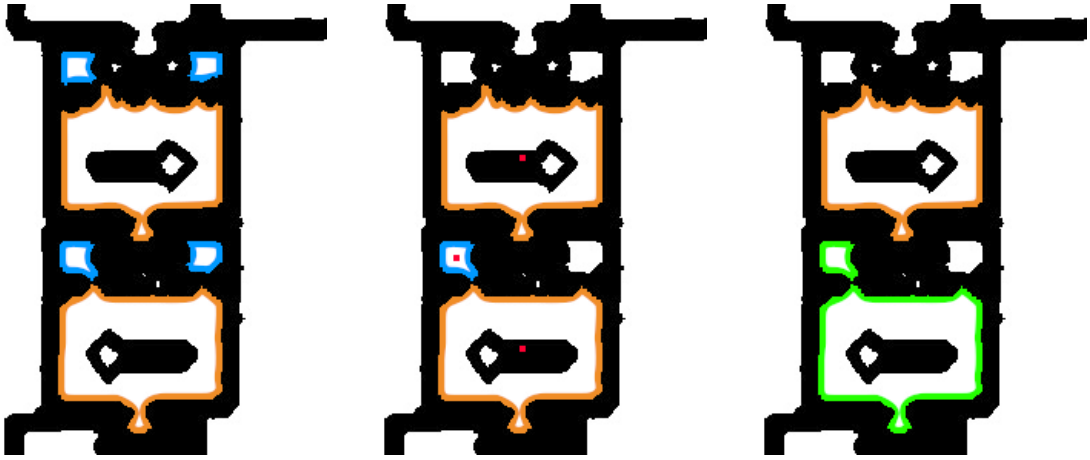
1. find contours;
2. reduce contours down to the "children";
3. filter contours to obtain the most tile alike ones;
4. get ROI and resize it to 32x64 px;
5. detect teeth in the ROI of all contours using the trained detector:
   (a) if no teeth were detected, rotate the image 90º clockwise and return to 1.;
6. check whether the two rules are met:
   (a) if no, rotate the image 90º clockwise and return to 1.;
   (b) if yes, the 0º position was found.

The previous procedure is repeated up to four times or until the 0º position is found, this way, in the worst case scenario, the four possible rotation angles (0º, 90º, 180º and 270º) are all checked. For instance, if a Tactode code is rotated 180º clockwise and assuming that the teeth detector works properly, the image would be rotated two times, going through the angles of 180º, 270º and 0º clockwise, and would stop after the second rotation which corresponds to an angle of 0º.

3.2.2.5 Filtering and Agglomerating Contours

At this point, the image is correctly oriented and the child contours have already been computed, but to work with them first it is required to perform some filtering to remove noisy contours such as very small ones, whose area are less than 1% of the area of the biggest contour, and to remove inner contours which are inside others.

Nevertheless, other problematic issues can appear. Figure 3.18a presents one problem that regularly occur which is the separation among the "body" of tiles (in orange) and their top "corners" (in blue). This happens especially when the code is bigger causing that the tiles look smaller in the photograph or when the photograph is simply zoomed out. This problem had to be solved because it would lead to obtain smaller bounding boxes for the respective tiles and, consequently, classification errors of the same could occur. The solution to tackle this problem was to aggregate top corners that were separated to their correspondent tiles.



(a) (b) (c)

Figure 3.18: The separation between the body and corners of tiles: (a) highlighted corners and bodies, (b) corner in focus, two nearest bodies and their centers, (c) corner assignment.

An approach based on colour thresholding - where the colour of the top corner would right-away assign itself to a tile of the same colour - is not the most adequate solution to this problem because it was enough that two tiles of the same colour were connected to confuse the whole procedure. Therefore, to tackle this problem a distance-based approach was taken in which, for each top corner, the distance of its bounding box center to the center of the bounding box of each tile body is calculated following (3.2), where $C = (x_C, y_C)$ represents the center of the corner and $B = (x_B, y_B)$ the center of the tile body.

$$Distance = \sqrt{(x_C - x_B)^2 + (y_C - y_B)^2} \tag{3.2}$$

After having computed the distances, some checks on the two nearest tile bodies are made to assess which one the corner belongs to. These checks are whether the corner center, in red in

Figure 3.18b, is above the body center and is inside two lateral limits corresponding to the lateral sides of the body. In case these two conditions are verified for one of the two candidate bodies, the corner is assigned to that body, as can be seen in Figure 3.18c. Otherwise, if the two conditions are not verified for any of the candidate bodies, the corner is assigned to the nearest. The assignment consists on the tile body bounding box being united with the corner bounding box, forming a bigger bounding box.

3.2.2.6 Positioning

At this stage, the remaining bounding boxes correspond to the Tactode tiles present in the image, as can be verified in Figure 3.19. These tiles have relative positions with each other that are important for building the Abstract Syntax Tree (AST) [64] that in turn the application will convert into a file that the target platform understands and can execute [1].

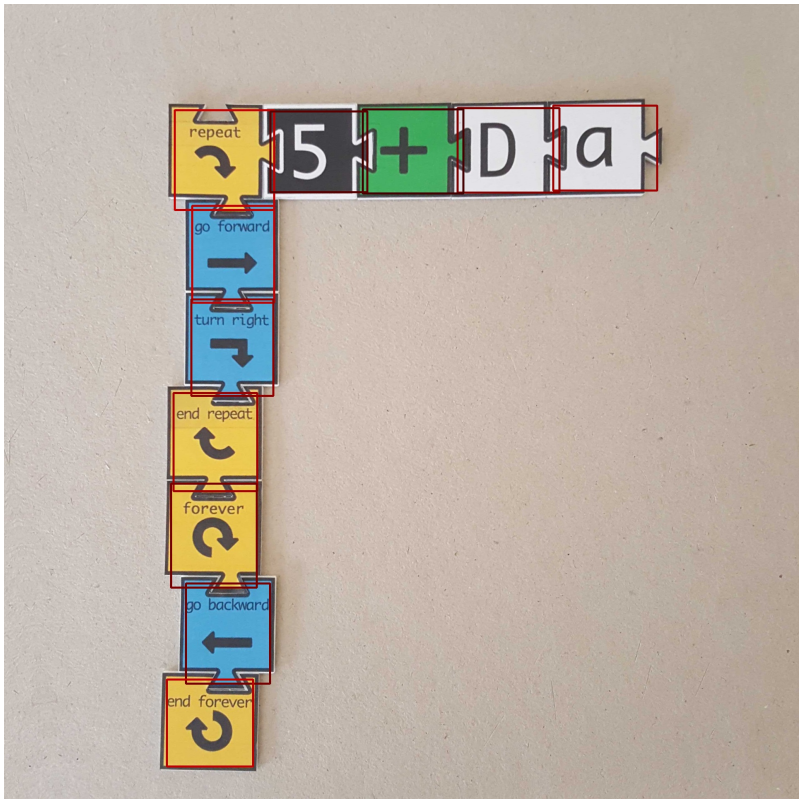


Figure 3.19: The bounding boxes around the tiles.

The approach chosen for representing the positions of the tiles in the image was the use of a matrix. In Algorithm 2 is demonstrated the procedure that builds the aforementioned matrix which for each position has a bounding rectangle that corresponds to a tile that will be further classified.

Algorithm 2: Tiles positioning

```
Result: boundRectsArr[][] containing well positioned bounding rectangles
sort boxesArr[] by ascending y order;
set tilesArr[][];
set row[];
set prevBox to undefined;
for each box in boxesArr[] do
    if prevBox == undefined then
        set prevBox to box;
    else
        if box.y > prevBox.y + prevBox.height / 4 then
            // new row of tiles found
            push prevBox to row[];
            sort row[] by ascending x order;
            push row[] to boundRectsArr[][];
            reset row[];
        else
            // same row
            push prevBox to row[];
        end
        if box.index == boxesArr.length - 1 then
            // last box
            push box to row[];
            sort row[] by ascending x order;
            push row[] to boundRectsArr[][];
        end
        set prevBox to box;
    end
end
```

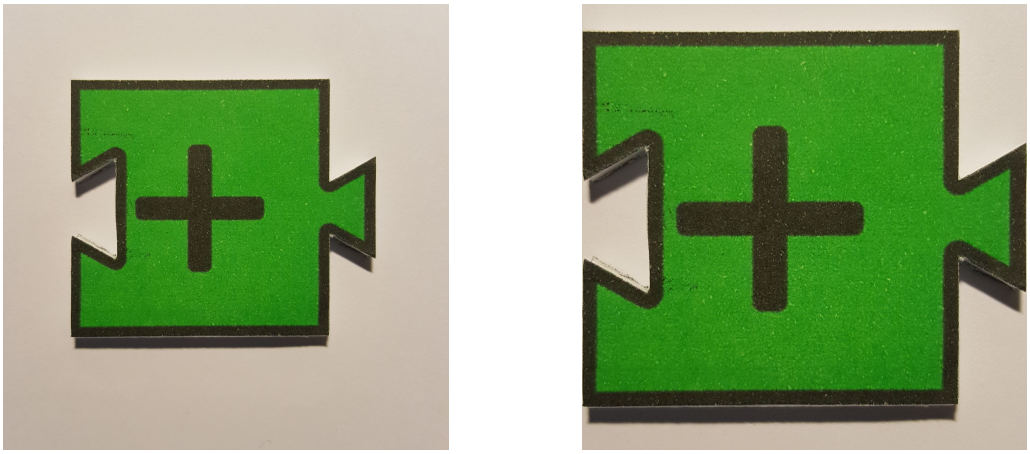
3.2.2.7 Regions Of Interest Classification

After the matrix being filled with correctly positioned bounding boxes, the final stage is the classification of corresponding tiles. The technique chosen to accomplish this was a also the use of HOG descriptors combined with a SVM classifier. To train the classifier, was assembled a training dataset of 300 images (samples) per tile, thus resulting on a total of 300 images/tile $\times$ 32 tiles = 9600 images.

Since 32 different tiles must be classified, a multiclass SVM is required. Thus, recurring to the training dataset already mentioned, a linear SVM (C = 0.1) was trained using HOG descriptors computed from each sample of the dataset. The HOG descriptor is calculated over a window size of 32x32 px, divided into cells of 8x8 px, with blocks composed by 2x2 cells (or 16x16 px) and a block stride of one cell (or 8x8 px). The gradients are signed and each histogram have nine bins, leading to a descriptors vector with a length of 324 per sample.

During the training, from each image sample, the ROI - part of the image containing the tile - is extracted using the bounding box, which in turn is adapted to a square shape (1:1 ratio) and resized

to 32x32 px (the window size of HOG), the descriptors vector is computed and then, is given to the SVM classifier. In addition, a 10-fold cross validation test was performed after the classifier training to assess the quality of the same and the result obtained was zero errors. Figure 3.20 presents two images of the same tile both related to the ROI extraction operation for training.



(a) (b)

Figure 3.20: Two images of the tile "addition" for training the classifier: image (a) is a photograph of the tile and image (b) is the respective ROI that is already shaped like a square.

With the model already trained, two generated files containing the configuration of HOG and the trained model are saved to be loaded and used later to classify the tiles. The classification procedure boils down to Algorithm 3, where the input is a matrix containing for each position a bounding rectangle and the result is a matrix holding in each position a named class represented by a string-typed variable, as is shown in Figure 3.21.

```
[
  [ 'repeat', '5', 'addition', 'upper_D', 'lower_a' ],
  [ 'forward' ],
  [ 'right' ],
  [ 'end_repeat' ],
  [ 'forever' ],
  [ 'backward' ],
  [ 'end_forever' ]
]
```

Figure 3.21: Resulting matrix with the classification labels.

Algorithm 3: Tiles classification

```
Result: tilesArr[][] containing the classified tiles
load HOG from fileHOG;
load SVM from fileSVM;
set tilesArr[][] and tilesRow[];
for each row[] in boundRectsArr[][] do
    for each boundRect in row[] do
        update boundRect to a square shape;
        get ROI from sourceImg using boundRect;
        resize ROI to 32x32 px;
        compute descriptors using HOG with ROI;
        predict label using SVM with descriptors;
        compute class using label;
        push class to tilesRow[];
    end
    push tilesRow[] to tilesArr[][];
    reset tilesRow[];
end
```

3.3 Integration Experiments

To assess the integration of the algorithm in the Tactode programming system, was developed a mini web application, adapted from [65]. This application was constructed in HTML, CSS and JavaScript, and was used the *OpenCV* JavaScript library instead of the aforementioned node package. The design of the web application can be seen in Figure 3.22. The user uploads an image of a Tactode code and click "Run", generating three images: the binary image, the image with only the parent and child contours, and the image with the ROIs before the classification stage.

Since the current version of *OpenCV* does not provide the necessary bindings for HOG and SVM, the program run on the web application does not possess the stages for rotation correction and tiles classification, because these directly depend on HOG and SVM. All the other stages work correctly and similarly to the Node.js version of the algorithm.

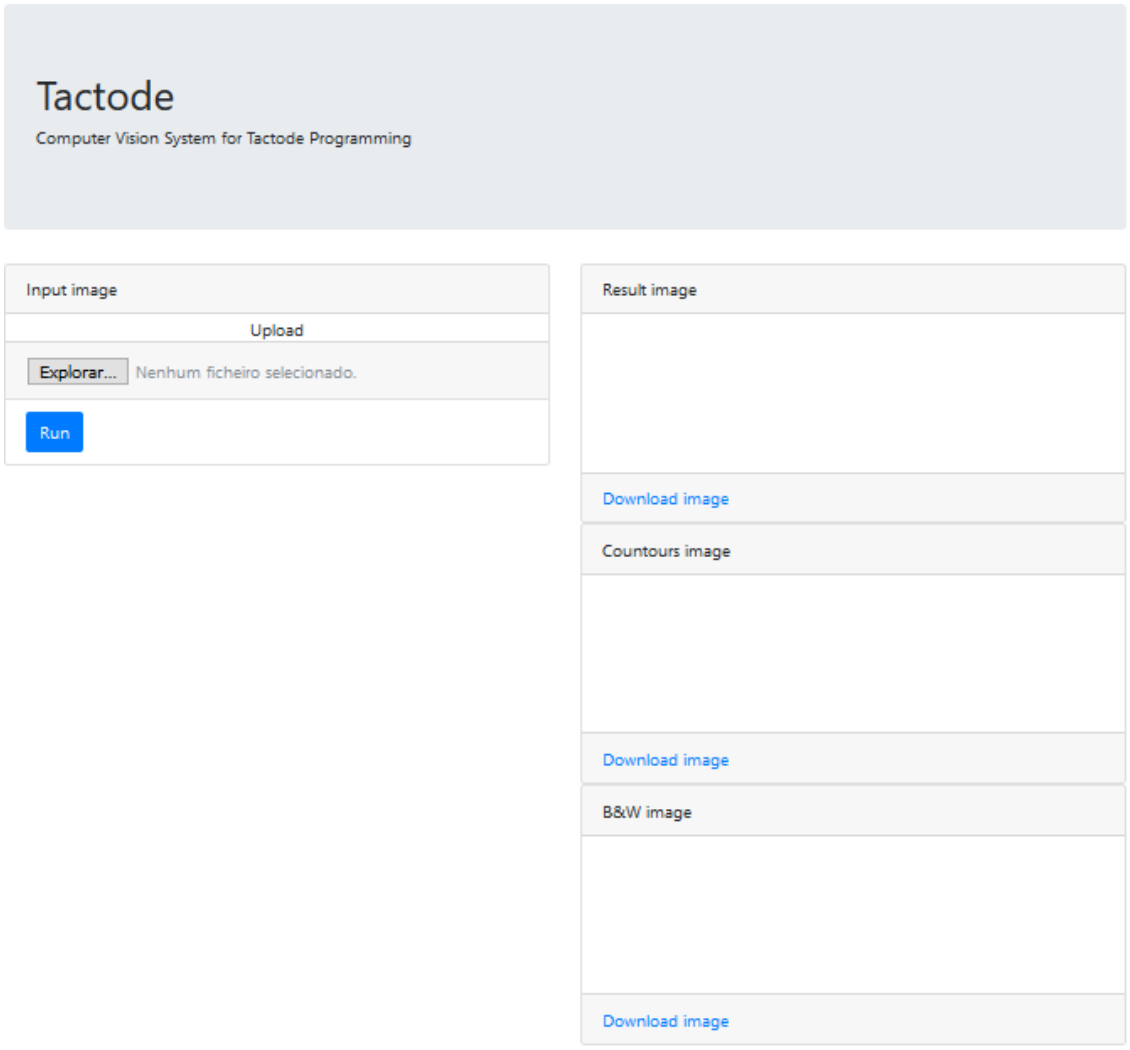


Figure 3.22: Tactode web application for integration experiments.

Chapter 4

Experimental Validation

This chapter presents the datasets on which the SVM classifiers were trained and the developed algorithm was tested to aid validating it. The range of resolutions, the number of images and tiles are some of the characteristics exposed about the datasets. Also, different classification methods that were explored are introduced later on this chapter.

4.1 Training Datasets

The detection of "teeth" of the tiles and the tile classification, as explained in sections 3.2.2.4 and 3.2.2.7, respectively, were accomplished with the utilisation of HOG and SVM. Therefore, for each of the aforementioned tasks, was created a dataset to train the SVM.

The two datasets are available online [1] and share some images which were obtained by recording for each tile a 10 second video at 30 fps and then, using a JavaScript program, each frame was extracted and counted as a tile sample, resulting in 300 images/tile $\times$ 32 tiles = 9600 images. This way is faster to gather training samples and also, each video was recorded so that the acquired images presented variance in terms of rotation and perspective. The tables 4.1 and 4.2 show the characteristics of the training datasets for "teeth" detection and tile classification, respectively.

Table 4.1: Specifications of the training dataset for tiles "teeth" detection.

Device	**Resolution (px)**	**Number of pixels (Mpx)**	**Type of sample**	**Number of images**
Samsung Galaxy S6	3840x2160	8.3	positive	2100
			negative	2430

[1] Tactode. Tactode Datasets. Available at http://fe.up.pt/asousa/tactode_datasets. Last accessed 2020-07-05.

Table 4.2: Specifications of the training dataset for tile classification.

Device	Resolution (px)	Number of pixels (Mpx)	Number of images
Samsung Galaxy S6	3840x2160	8.3	9600

4.2 Test Datasets

To validate the implemented software solution described in Section 3.2.2 were built two datasets for testing: a *small* dataset and a *big* dataset. Both datasets are available online [1] and are made of images of different Tactode codes with multiples resolutions obtained by using three devices (two smartphones and one tablet). This way, if the computer vision algorithm finishes the tests of these datasets successfully, it can be said that the computer vision algorithm works on multiples devices and at multiple resolutions. The devices and their camera resolutions are presented below in Table 4.3 and Table 4.4 presents the specifications of the laptop where all tests were run.

Table 4.3: Camera resolutions of the devices used.

Device	Camera resolution (Mpx)
Samsung Galaxy S6	16
Huawei P Smart+	10
Samsung Galaxy Tab S	8

Table 4.4: Computer specifications where the tests were executed.

Name	OS	CPU	RAM
HP Notebook	Ubuntu 18.04 LTS	Intel i7 Quad-core @ 2.4GHz	4GB

4.2.1 *Small* Dataset

The *small* dataset is made up by 288 photographs over six distinct codes having a total of 2784 tiles. In Appendix A, the figures A.1, A.2, A.3, A.4, A.5 and A.6 show the six codes that make up this dataset. For each code in each resolution, **four** images with some variations were taken: a frontal and non-rotated image, a frontal and rotated by +10º image, a frontal and rotated by -10º image and an image with perspective from the bottom of the code. The rotation angles mentioned above are an approximation.

The distribution of resolutions and number of images and tiles per device is shown in Table 4.5. Each resolution row of the table have a number of images equivalent to 6 $codes \times 4$ $images$ and a number of tiles equal to $(11+5+4+11+19+8)$ $tiles \times 4$ $images$.

This dataset was created to validate whether the algorithm was able to recognise all the tiles and also, to compare the classification accuracy of HOG and SVM with other methods, such as

Template Matching and methods based on the detection and description of image features, that will be mentioned later.

Table 4.5: The resolutions present in the *small* dataset and the respective number of images and tiles.

Device	**Resolution (px)**	**Number of pixels (Mpx)**	**Number of images**	**Number of tiles**	
Samsung Galaxy S6	5312x2988	16	24	232	
	3984x2988	12	24	232	
	2976x2976	8.9	24	232	
	3264x2448	8	24	232	
	3264x1836	6	24	232	
Huawei P Smart+	3648x2736	10	24	232	
Samsung Galaxy Tab S	3264x2448	8	24	232	
	3264x1836	6	24	232	
	2560x1920	4.9	24	232	
	2560x1440	3.7	24	232	
	2048x1536	3.1	24	232	
	2048x1152	2.4	24	232	
			288	2784	**Total**

4.2.2 *Big* Dataset

The *big* dataset is made up by 12000 images over two distinct codes having a total of 96000 tiles. The two codes can be seen in figures A.7 and A.8 from Appendix A. The images were obtained by extracting the frames from 10 second videos at 30 fps that were taped for each code. For each code, were recorded videos with rotation angles of 0º, 90º, 180º and 270º clockwise (**one** video per angle), resulting in test images rotated by the same angles. This approach of recording videos and then, using a JS program, extracting the frames is an easy and fast way of generating multiple test images. Beside that, each video was recorded with some minimal variations of perspective and rotation.

The distribution of resolutions, rotation angles and number of images and tiles per device is shown in Table 4.6. Each row of the table related to a rotation angle (and the respective resolution) have a number of images equivalent to 2 $codes \times 300$ $images$ and a number of tiles equal to $(11+5)$ $tiles \times 300$ $images$.

This dataset aims to completely test the algorithm that was developed to verify, whether the segmentation is well executed, the algorithm is rotation-invariant and the tiles classification is carried out successfully.

Table 4.6: The resolutions present in the *big* dataset with the respective rotations and number of images and tiles.

Device	Resolution (px)	Number of pixels (Mpx)	Rotation angle (clockwise)	Number of images	Number of tiles	
Samsung Galaxy S6	3840x2160	8.3	0º	600	4800	
			90º	600	4800	
			180º	600	4800	
			270º	600	4800	
	2560x1440	3.7	0º	600	4800	
			90º	600	4800	
			180º	600	4800	
			270º	600	4800	
	1920x1080	2.1	0º	600	4800	
			90º	600	4800	
			180º	600	4800	
			270º	600	4800	
Huawei P Smart+	1920x1080	2.1	0º	600	4800	
			90º	600	4800	
			180º	600	4800	
			270º	600	4800	
Samsung Galaxy Tab S	1920x1080	2.1	0º	600	4800	
			90º	600	4800	
			180º	600	4800	
			270º	600	4800	
				12000	96000	**Total**

4.3 Baseline Classification Methods

The baseline methods used to classify the Tactode tiles were Template Matching (presented in Section 2.1.2) and Feature Detectors and Descriptors, namely, ORB, SURF, SIFT and BRISK (presented in Section 2.2). In order to work with these, was generated a template image with a resolution of 350x350 px per tile, giving a total of 32 templates. In Figure 4.1 are presented three template examples.

For any method, the algorithm is exactly the same, excluding the classification stage, where it has slight changes comparing to the one presented in Section 3.2.2.7. It starts by adjusting the bounding rectangle of each tile to a square shape, followed by a resizing to 350x350 px, obtaining the ROI. After this, the remaining procedure steps are distinct with respect to the method being used.

Regarding the Template Matching, the remaining steps are based on performing a template match between the ROI and every template. Then the classification of the tile (ROI) is determined by the template with which the template matching operation between them resulted on the highest - in absolute value - maximum or minimum value (depending on the matching mode), meaning that they have the biggest correlation. The maximum or minimum values are considered according



(a) (b) (c)

Figure 4.1: Three templates of the 32 template set: (a), (b) and (c) are the templates for the "turn left", "end forever" and "pen down" tiles, respectively.

to whether the template matching mode being used is *CCOEFF* or *SQDIFF*, respectively [2] [33].

With respect to the methods based on features detection and description, they detect the keypoints and compute the descriptors of the target ROI and the same is executed for every template. After, a feature matcher matches the descriptors of the ROI with the descriptors of each template. Then, the outliers are filtered recurring to Random Sample Consensus (RANSAC) with a maximum reprojection error threshold of 5, a 99.5% confidence level and a maximum iterations number of 2000. Finally, the template that results in more inliers is the one who classifies the respective tile. The feature detectors and descriptors were configured in order to the numbers of keypoints detected was similar among them. These settings are specified in Table 4.7.

Table 4.7: Settings of the methods for feature detection and description.

Method	**Settings**
ORB	*maxFeatures* : 600
BRISK	*threshold* : 10
SURF	*hessianThreshold* : 50
SIFT	*nFeatures* : 600

The feature matcher chosen was Brute-Force with cross-check along with an operation of *knn* matching with k=1. *OpenCV* explicitly stipulates that this combination produces better results and is an alternative to the ratio test [3] [33] presented in [35]. Also the norms chosen for the matcher were *L1* for SIFT and SURF, and *Hamming* for ORB and BRISK, complying to what is specified in *OpenCV* [3] [33].

[2] OpenCV. Template Matching. Available at https://docs.opencv.org/4.4.0/d8/dd1/tutorial_js_template_matching.html. Last accessed 2020-07-05.

[3] OpenCV. BFMatcher Class Reference. Available at https://docs.opencv.org/4.4.0/d3/da1/classcv_1_1BFMatcher.html. Last accessed 2020-07-05.

Chapter 5

Results

In this chapter the results of the tests taken on the datasets are presented and interpreted. Initially are shown the results of the *small* dataset where various classification methods are compared. Then, the results of the *big* dataset are presented, for which only the HOG and SVM classification method was considered as way of fully test the developed vision-based system for this dissertation.

5.1 Preliminary Information

This section introduces some preliminary aspects about the obtained results of the tests, such as the errors that could come up while testing, and the assessment metrics that help evaluate the performance of the algorithm.

With respect to the tests applied on both datasets, these could result in three different errors:

1. Rotation error: if the rotation of the Tactode code was not found, resulting in a rotated code.
2. Segmentation error: whether the border of any tile "broke" or the agglomeration and filtering of contours was not successfully accomplished.
3. Classification error: if the tiles were wrongly classified.

The rotation and segmentation errors did not occur, appearing only classification errors. Therefore, regarding the pre-classification stages of the developed algorithm, one can say that they demonstrate 100% accuracy on the test datasets.

Concerning the classification errors, the measure to assess the quality of the method that classified the tiles is defined by (5.1), where T and MT represent the number of tiles and the number of misclassified tiles, respectively.

$$Accuracy = \frac{T - MT}{T} \times 100\% \tag{5.1}$$

In addition, the Execution Time Mean Value (ETMV) and Execution Time Standard Deviation (ETSD) were also measured. These time metrics, expressed in seconds, are specified in (5.2) and

(5.3), where N is the number of executions, t_i is the execution time at instance i, μ is the mean value and σ is the standard deviation.

$$\mu = \frac{\sum_{t=1}^{N} t_i}{N} \tag{5.2}$$

$$\sigma = \sqrt{\frac{\sum_{t=1}^{N} (t_i - \mu)^2}{N}} \tag{5.3}$$

Lastly, a metric as chosen to evaluate the performance of the method in terms of percentage of images with errors. This metric is defined by (5.4), where I and EI are the number of images and the number of images with errors, respectively.

$$Correctness = \frac{I - EI}{I} \times 100\% \tag{5.4}$$

5.2 Results of the Tests Executed on the *Small* Dataset

This section is dedicated to show the results of the tests run on the *small* dataset. It is important to remember that this dataset is composed by 288 test images and 2784 tiles. The overall results of the tests with different classification methods on this dataset are presented in Table 5.1.

Table 5.1: Resulting accuracy, ETMV and ETSD from the tests made with various classification methods on the *small* dataset.

Method	**Misclassified tiles**	**Accuracy (%)**	**ETMV (s)**	**ETSD (s)**
HOG & SVM	0	100	0.960	0.305
SURF	31	98.89	36.648	18.347
ORB	127	95.44	14.413	6.417
BRISK	193	93.07	20.087	9.350
SIFT	364	86.93	26.536	12.435
TM (CCOEFF)	514	81.54	3.584	1.513
TM (SQDIFF)	852	69.40	3.867	1.672

After analysing the table above, it is possible to conclude that the classification method based on HOG and SVM clearly outperforms all the other methods in terms of accuracy (**100%**) and time of execution. SURF is ranked second for accuracy purposes, but it has the worst performance when referring to execution time. Typically, SURF brings less computational cost than SIFT [49], however in this case, the outliers filtering is slowing down the SURF version of the classification stage, since this detects more keypoints than SIFT, thus have more outliers. Also, the methods that consist on the detection and description of features - SURF, SIFT, ORB and BRISK - have come up with a better accuracy than any of the Template Matching (TM) versions.

Table 5.2 presents the correctness level of the several algorithms in terms of number of images with classification errors. From the table below one may conclude that, HOG and SVM achieved a correctness level of **100%**, as expected since this method did not presented any classification error, and SURF was the second best method with only 24 images introducing errors, giving a 91.67% correctness level.

Table 5.2: Resulting correctness level from the tests made with various classification methods on the *small* dataset.

Method	**Images with errors**	**Correctness (%)**
HOG & SVM	0	100
SURF	24	91.67
BRISK	95	67.01
ORB	112	61.11
SIFT	147	48.96
TM (CCOEFF)	165	42.71
TM (SQDIFF)	209	27.43

To appraise the performance of the methods in different image resolutions, tables 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 present the accuracy of them distributed by the range of resolutions. Regarding these tables, it is important to notice that, for the resolutions of 6 and 8 Mpx, the number of test tiles is doubled (468 tiles) compared to the other resolutions, due to two of the devices used have both of these resolutions.

Table 5.3: Results of the tests on the *small* dataset with HOG and SVM for each resolution.

HOG & SVM		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	0	100
12	0	100
10	0	100
8.9	0	100
8	0	100
6	0	100
4.9	0	100
3.7	0	100
3.1	0	100
2.4	0	100

Table 5.4: Results of the tests on the *small* dataset with SURF for each resolution.

SURF		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	3	98.71
12	2	99.14
10	2	99.14
8.9	1	99.57
8	4	99.14
6	8	98.28
4.9	1	99.57
3.7	7	96.98
3.1	0	100
2.4	3	98.71

Table 5.5: Results of the tests on the *small* dataset with SIFT for each resolution.

SIFT		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	30	87.07
12	30	87.07
10	25	89.22
8.9	27	88.36
8	59	87.28
6	63	86.42
4.9	28	87.93
3.7	35	84.91
3.1	31	86.64
2.4	36	84.48

Table 5.6: Results of the tests on the *small* dataset with BRISK for each resolution.

BRISK		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	15	93.53
12	11	95.26
10	11	95.26
8.9	15	93.53
8	23	95.04
6	45	90.30
4.9	11	95.26
3.7	25	89.22
3.1	14	93.97
2.4	23	90.09

Table 5.7: Results of the tests on the *small* dataset with ORB for each resolution.

ORB		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	10	95.69
12	11	95.26
10	15	93.53
8.9	10	95.69
8	18	96.12
6	26	94.40
4.9	9	96.12
3.7	10	95.69
3.1	8	96.55
2.4	10	95.69

Table 5.8: Results of the tests on the *small* dataset with Template Matching (*CCOEFF*) for each resolution.

Template Matching - CCOEFF		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	34	85.34
12	36	84.48
10	77	66.81
8.9	27	88.36
8	60	87.07
6	85	81.68
4.9	25	89.22
3.7	37	84.05
3.1	56	75.86
2.4	77	66.81

Table 5.9: Results of the tests on the *small* dataset with Template Matching (*SQDIFF*) for each resolution.

Template Matching - SQDIFF		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
16	90	61.21
12	83	64.22
10	86	62.93
8.9	83	64.22
8	131	71.77
6	146	68.53
4.9	33	85.78
3.7	43	81.47
3.1	73	68.53
2.4	84	63.79

From the preceding tables, it is not possible to conclude anything whereas there is no notorious changes on the accuracy results over the possible resolutions.

Next, tables 5.10, 5.11, 5.12, 5.13, 5.14, 5.15 and 5.16 assess the tiles predictions resulting from the tests run on the *small* dataset. The evaluation is made recurring to some metrics namely, Precision, Recall, F1-score and their components, true positives (TP), false positives (FP) and false negatives (FN). Additionally, in figures B.1, B.2, B.3, B.4, B.5, B.6 and B.7 from Appendix B, are the confusion matrices of the tiles for each classification method tested on the *small* dataset. The columns of the confusions matrices comprise the truth classes and its false negatives, and the rows include the predicted classes and its false positives. The diagonal of the matrices is where the true positives are.

Table 5.10: Evaluation of the tiles classification for the tests with HOG and SVM on the *small* dataset.

HOG & SVM						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	0	0	1.00	1.00	1.00
end repeat	240	0	0	1.00	1.00	1.00
forever	96	0	0	1.00	1.00	1.00
repeat	240	0	0	1.00	1.00	1.00
flag	96	0	0	1.00	1.00	1.00
lower a	48	0	0	1.00	1.00	1.00
lower d	48	0	0	1.00	1.00	1.00
lower e	48	0	0	1.00	1.00	1.00
lower i	48	0	0	1.00	1.00	1.00
lower s	96	0	0	1.00	1.00	1.00
upper D	48	0	0	1.00	1.00	1.00
backward	240	0	0	1.00	1.00	1.00
forward	240	0	0	1.00	1.00	1.00
left	144	0	0	1.00	1.00	1.00
right	192	0	0	1.00	1.00	1.00
0	48	0	0	1.00	1.00	1.00
1	48	0	0	1.00	1.00	1.00
2	48	0	0	1.00	1.00	1.00
3	48	0	0	1.00	1.00	1.00
4	48	0	0	1.00	1.00	1.00
5	48	0	0	1.00	1.00	1.00
6	48	0	0	1.00	1.00	1.00
7	48	0	0	1.00	1.00	1.00

Continued on next page

Table 5.10 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
8	48	0	0	1.00	1.00	1.00
9	48	0	0	1.00	1.00	1.00
addition	48	0	0	1.00	1.00	1.00
division	96	0	0	1.00	1.00	1.00
answer	48	0	0	1.00	1.00	1.00
question	48	0	0	1.00	1.00	1.00
erase	48	0	0	1.00	1.00	1.00
pen down	48	0	0	1.00	1.00	1.00
pen up	48	0	0	1.00	1.00	1.00

Table 5.11: Evaluation of the tiles classification for the tests with SURF on the *small* dataset.

SURF						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	1	0	0.989	1.00	0.994
end repeat	240	3	0	0.987	1.00	0.993
forever	96	1	0	0.989	1.00	0.994
repeat	240	3	0	0.987	1.00	0.993
flag	96	1	0	0.989	1.00	0.994
lower a	48	1	0	0.979	1.00	0.989
lower d	47	0	1	1.00	0.979	0.989
lower e	47	2	1	0.959	0.979	0.968
lower i	46	0	2	1.00	0.958	0.978
lower s	84	0	12	1.00	0.875	0.933
upper D	48	0	0	1.00	1.00	1.00
backward	240	6	0	0.975	1.00	0.987
forward	240	0	0	1.00	1.00	1.00
left	144	1	0	0.993	1.00	0.996
right	192	1	0	0.994	1.00	0.996
0	48	1	0	0.979	1.00	0.989
1	48	0	0	1.00	1.00	1.00
2	48	0	0	1.00	1.00	1.00
3	48	0	0	1.00	1.00	1.00
4	48	0	0	1.00	1.00	1.00
5	48	0	0	1.00	1.00	1.00

Continued on next page

Table 5.11 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
6	46	0	2	1.00	0.958	0.978
7	48	0	0	1.00	1.00	1.00
8	48	0	0	1.00	1.00	1.00
9	47	2	1	0.959	0.979	0.968
addition	44	2	4	0.956	0.916	0.935
division	88	0	8	1.00	0.916	0.956
answer	48	6	0	0.888	1.00	0.940
question	48	0	0	1.00	1.00	1.00
erase	48	0	0	1.00	1.00	1.00
pen down	48	0	0	1.00	1.00	1.00
pen up	48	0	0	1.00	1.00	1.00

Table 5.12: Evaluation of the tiles classification for the tests with SIFT on the *small* dataset.

SIFT						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	33	0	0.744	1.00	0.853
end repeat	240	29	0	0.892	1.00	0.942
forever	96	10	0	0.905	1.00	0.950
repeat	240	19	0	0.926	1.00	0.961
flag	95	18	1	0.840	0.989	0.908
lower a	42	11	6	0.792	0.875	0.831
lower d	37	10	11	0.787	0.770	0.778
lower e	29	22	19	0.568	0.604	0.585
lower i	22	1	26	0.956	0.458	0.619
lower s	20	3	76	0.869	0.208	0.335
upper D	3	5	45	0.375	0.0620	0.106
backward	240	22	0	0.916	1.00	0.956
forward	240	11	0	0.956	1.00	0.977
left	144	4	0	0.972	1.00	0.985
right	192	6	0	0.969	1.00	0.984
0	42	5	6	0.893	0.875	0.883
1	47	11	1	0.810	0.979	0.886
2	48	9	0	0.842	1.00	0.914
3	48	20	0	0.705	1.00	0.826

Continued on next page

Table 5.12 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
4	48	5	0	0.905	1.00	0.950
5	48	3	0	0.941	1.00	0.969
6	44	5	4	0.897	0.916	0.906
7	40	3	8	0.930	0.833	0.878
8	35	6	13	0.853	0.729	0.786
9	27	0	21	1.00	0.562	0.719
addition	10	10	38	0.500	0.208	0.293
division	12	1	84	0.923	0.125	0.220
answer	48	60	0	0.444	1.00	0.614
question	48	3	0	0.941	1.00	0.969
erase	47	15	1	0.758	0.979	0.854
pen down	44	0	4	1.00	0.916	0.956
pen up	48	4	0	0.923	1.00	0.959

Table 5.13: Evaluation of the tiles classification for the tests with BRISK on the *small* dataset.

BRISK						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	7	0	0.932	1.00	0.964
end repeat	240	14	0	0.944	1.00	0.971
forever	96	6	0	0.941	1.00	0.969
repeat	240	5	0	0.979	1.00	0.989
flag	96	2	0	0.979	1.00	0.989
lower a	46	7	2	0.867	0.958	0.910
lower d	45	2	3	0.957	0.937	0.946
lower e	48	3	0	0.941	1.00	0.969
lower i	13	1	35	0.928	0.270	0.418
lower s	51	2	45	0.962	0.531	0.684
upper D	32	2	16	0.941	0.666	0.779
backward	240	10	0	0.960	1.00	0.979
forward	240	2	0	0.991	1.00	0.995
left	144	4	0	0.972	1.00	0.985
right	192	3	0	0.984	1.00	0.991
0	46	14	2	0.766	0.958	0.851
1	39	6	9	0.866	0.812	0.838

Continued on next page

Table 5.13 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
2	37	6	11	0.860	0.770	0.812
3	46	8	2	0.851	0.958	0.901
4	48	10	0	0.827	1.00	0.905
5	44	3	4	0.936	0.916	0.925
6	47	5	1	0.903	0.979	0.939
7	30	22	18	0.576	0.625	0.599
8	48	2	0	0.960	1.00	0.979
9	46	3	2	0.938	0.958	0.947
addition	41	4	7	0.911	0.854	0.881
division	60	0	36	1.00	0.625	0.769
answer	48	26	0	0.648	1.00	0.786
question	48	5	0	0.905	1.00	0.950
erase	48	3	0	0.941	1.00	0.969
pen down	48	3	0	0.941	1.00	0.969
pen up	48	3	0	0.941	1.00	0.969

Table 5.14: Evaluation of the tiles classification for the tests with ORB on the *small* dataset.

ORB						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	6	0	0.941	1.00	0.969
end repeat	240	3	0	0.987	1.00	0.993
forever	96	3	0	0.969	1.00	0.984
repeat	240	1	0	0.995	1.00	0.997
flag	96	2	0	0.979	1.00	0.989
lower a	47	5	1	0.903	0.979	0.939
lower d	45	8	3	0.849	0.937	0.890
lower e	48	5	0	0.905	1.00	0.950
lower i	46	0	2	1.00	0.958	0.978
lower s	96	3	0	0.969	1.00	0.984
upper D	45	1	3	0.978	0.937	0.957
backward	240	2	0	0.991	1.00	0.995
forward	240	1	0	0.995	1.00	0.997
left	144	2	0	0.986	1.00	0.992
right	192	1	0	0.994	1.00	0.996

Continued on next page

Table 5.14 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
0	47	2	1	0.959	0.979	0.968
1	48	1	0	0.979	1.00	0.989
2	45	0	3	1.00	0.937	0.967
3	48	3	0	0.941	1.00	0.969
4	48	3	0	0.941	1.00	0.969
5	46	2	2	0.958	0.958	0.957
6	48	3	0	0.941	1.00	0.969
7	45	1	3	0.978	0.937	0.957
8	48	0	0	1.00	1.00	1.00
9	48	1	0	0.979	1.00	0.989
addition	36	0	12	1.00	0.750	0.857
division	0	0	96	-	0.00	-
answer	48	40	0	0.545	1.00	0.705
question	48	27	0	0.640	1.00	0.780
erase	48	1	0	0.979	1.00	0.989
pen down	47	0	1	1.00	0.979	0.989
pen up	48	0	0	1.00	1.00	1.00

Table 5.15: Evaluation of the tiles classification for the tests with Template Matching (*CCOEFF*) on the *small* dataset.

Template Matching - CCOEFF						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	96	21	0	0.820	1.00	0.901
end repeat	194	0	46	1.00	0.808	0.893
forever	80	0	16	1.00	0.833	0.908
repeat	240	7	0	0.971	1.00	0.985
flag	73	0	23	1.00	0.760	0.863
lower a	35	57	13	0.380	0.729	0.499
lower d	48	43	0	0.527	1.00	0.690
lower e	47	41	1	0.534	0.979	0.691
lower i	48	138	0	0.258	1.00	0.410
lower s	81	4	15	0.952	0.843	0.894
upper D	48	3	0	0.941	1.00	0.969
backward	205	90	35	0.694	0.854	0.765

Continued on next page

Table 5.15 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
forward	187	11	53	0.944	0.779	0.853
left	133	0	11	1.00	0.923	0.959
right	188	0	4	1.00	0.979	0.989
0	48	1	0	0.979	1.00	0.989
1	48	5	0	0.905	1.00	0.950
2	46	0	2	1.00	0.958	0.978
3	19	0	29	1.00	0.395	0.566
4	45	0	3	1.00	0.937	0.967
5	48	31	0	0.607	1.00	0.755
6	47	2	1	0.959	0.979	0.968
7	48	2	0	0.960	1.00	0.979
8	47	0	1	1.00	0.979	0.989
9	48	3	0	0.941	1.00	0.969
addition	37	35	11	0.513	0.770	0.615
division	61	11	35	0.847	0.635	0.725
answer	4	6	44	0.400	0.0830	0.137
question	12	3	36	0.800	0.250	0.380
erase	9	0	39	1.00	0.187	0.315
pen down	0	0	48	-	0.00	-
pen up	0	0	48	-	0.00	-

Table 5.16: Evaluation of the tiles classification for the tests with Template Matching (*SQDIFF*) on the *small* dataset.

Template Matching - SQDIFF						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	95	53	1	0.641	0.989	0.777
end repeat	171	0	69	1.00	0.712	0.831
forever	76	0	20	1.00	0.791	0.883
repeat	224	39	16	0.851	0.933	0.890
flag	78	3	18	0.962	0.812	0.880
lower a	48	58	0	0.452	1.00	0.622
lower d	12	0	36	1.00	0.250	0.400
lower e	44	4	4	0.916	0.916	0.916
lower i	48	80	0	0.375	1.00	0.545

Continued on next page

Table 5.16 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
lower s	74	5	22	0.936	0.770	0.844
upper D	45	0	3	1.00	0.937	0.967
backward	118	36	122	0.766	0.491	0.598
forward	123	55	117	0.691	0.512	0.588
left	83	129	61	0.391	0.576	0.465
right	124	0	68	1.00	0.645	0.784
0	45	5	3	0.900	0.937	0.918
1	48	0	0	1.00	1.00	1.00
2	45	10	3	0.818	0.937	0.873
3	46	1	2	0.978	0.958	0.967
4	45	0	3	1.00	0.937	0.967
5	46	1	2	0.978	0.958	0.967
6	46	0	2	1.00	0.958	0.978
7	38	0	10	1.00	0.791	0.883
8	47	4	1	0.921	0.979	0.949
9	41	0	7	1.00	0.854	0.921
addition	29	66	19	0.305	0.604	0.405
division	30	1	66	0.967	0.312	0.471
answer	14	188	34	0.0690	0.291	0.111
question	12	106	36	0.101	0.250	0.143
erase	5	0	43	1.00	0.104	0.188
pen down	0	1	48	0.00	0.00	-
pen up	32	7	16	0.820	0.666	0.735

From the Table 5.10 and the confusion matrix shown in Figure B.1, can be observed that, for HOG and SVM, none of the tiles have false positives nor false negatives, resulting on a precision, recall and F1-score of 1.00 for all of them, as expected, since there was no misclassifications.

The SURF method was the one with less wrong predictions among the methods of features detection and description. For this method, Table 5.11 and Figure B.2 shows that, the tile with more false negatives, thus lower recall, was "lower s" presenting 12 false negatives and a recall of 0.875. The tiles "backward" and "answer" were the ones with more false positives (6), although "answer" has a worse precision (0.888) due to having less true positives - 48 against 240 of the tile "backward".

The *CCOEFF* is the best version of the Template Matching method. According to Table 5.15 and Figure B.6, "lower i" was the tile that generated more false positives (138), resulting on a

0.258 precision metric and, "forward" was the tile with more false negatives (53), thus having a recall of 0.779.

5.3 Results of the Tests Executed on the *Big* Dataset

This section presents the results of the tests using the original classification method (HOG and SVM) to recognise the tiles. These tests were executed on the *big* dataset which is composed by 12000 test images and 96000 tiles. Table 5.17 shows the test results with the accuracy, ETMV and ETSD that were obtained.

Table 5.17: Resulting accuracy, ETMV and ETSD from the tests made with HOG and SVM on the *big* dataset.

Method	Misclassified tiles	Accuracy (%)	ETMV (s)	ETSD (s)
HOG & SVM	12	99.99	0.730	0.179

From the table above, one can say that the classification stage of the algorithm is very robust, reaching an accuracy of **99.99%** along with an average execution time under one second.

Table 5.18 presents the correctness level of the algorithm in terms of number of images with classification errors. As can be seen in the table below, the algorithm achieved a **99.90%** correctness level.

Table 5.18: Resulting correctness level from the tests made with HOG and SVM on the *big* dataset.

Method	Images with errors	Correctness (%)
HOG & SVM	12	99.90

The resulting accuracy obtained for this method in the resolution range of the *big* dataset are presented in Table 5.19. It is important to note that the resolutions of 8.3 and 3.7 Mpx contain 19200 tiles each, and the 2.1 Mpx resolution contains 57600 tiles.

Table 5.19: Results of the tests on the *big* dataset with HOG and SVM for each resolution.

HOG & SVM		
Resolution (Mpx)	**Misclassified tiles**	**Accuracy (%)**
8.3	0	100
3.7	0	100
2.1	12	99.98

From the previous table, it is noticeable that the HOG and SVM based classification method behaves worse when the resolution decreases, since it demonstrates having misclassified tiles only at the lowest resolution (2.1 Mpx).

Table 5.20 shows the evaluation of the tiles classification using the same aforementioned metrics. Also, in Figure B.8 from Appendix B, is presented the confusion matrix of the tiles resulting from the tests run with HOG and SVM on the *big* dataset. After analysing these two, one can

conclude that the tiles that present more false positives and false negatives are "end forever" and "end repeat", respectively. In fact, according to the confusion matrix, these tiles are confused with each other with this method.

It is important to note that these tiles could generate even more classification errors if the modifications made to the "end forever" and "forever" tiles, mentioned in Section 3.1, had not been realised.

Table 5.20: Evaluation of the tiles classification for the tests with HOG and SVM on the *big* dataset.

HOG & SVM						
Tile	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-score**
end forever	6000	10	0	0.998	1.00	0.998
end repeat	11990	0	10	1.00	0.999	0.999
forever	5998	0	2	1.00	0.999	0.999
repeat	12000	0	0	1.00	1.00	1.00
flag	0	0	0	-	-	-
lower a	6000	0	0	1.00	1.00	1.00
lower d	0	0	0	-	-	-
lower e	0	2	0	0.00	-	-
lower i	0	0	0	-	-	-
lower s	0	0	0	-	-	-
upper D	6000	0	0	1.00	1.00	1.00
backward	12000	0	0	1.00	1.00	1.00
forward	6000	0	0	1.00	1.00	1.00
left	6000	0	0	1.00	1.00	1.00
right	6000	0	0	1.00	1.00	1.00
0	0	0	0	-	-	-
1	0	0	0	-	-	-
2	0	0	0	-	-	-
3	0	0	0	-	-	-
4	6000	0	0	1.00	1.00	1.00
5	6000	0	0	1.00	1.00	1.00
6	0	0	0	-	-	-
7	0	0	0	-	-	-
8	0	0	0	-	-	-
9	0	0	0	-	-	-
addition	6000	0	0	1.00	1.00	1.00
division	0	0	0	-	-	-

Continued on next page

Table 5.20 – *Continued from previous page*

Tile	TP	FP	FN	Precision	Recall	F1-score
answer	0	0	0	-	-	-
question	0	0	0	-	-	-
erase	0	0	0	-	-	-
pen down	0	0	0	-	-	-
pen up	0	0	0	-	-	-

5.4 Final Considerations

In this chapter were presented the results obtained for the tests that were made over the *small* and *big* datasets. In both cases, were not verified any rotation error nor segmentation error.

In the *small* dataset, the tests resulted on the HOG and SVM being the best tile recognition method attaining a classification accuracy of **100%**. Also this method sits on top of the execution time rank with a run time around one second.

Concerning the tests on the *big* dataset, the only method being tested was HOG and SVM which accomplished an accuracy of **99.99%** and whose misclassifications only occurred at the lowest of the resolutions, being that "end forever" was the tile with the lowest F1-score with a value of **0.998**.

These results demonstrate that the proposed computer vision system is capable of recognise the Tactode tiles and codes.

Chapter 6

Conclusions

This dissertation intended to develop a computer vision algorithm that could detect and distinguish the Tactode tiles without depending on ArUco markers. To do so, new tiles were constructed followed by the software development for the algorithm.

The result is a set of 3D-printed and ArUco-less Tactode tiles, and a JavaScript program based on *OpenCV* that receives a photograph of a Tactode code and, after processing it, returns a matrix that, for each position have a string-typed variable representing the class of the respective tile and also, the tiles are located in the matrix as they are in the photo. In addition, two training datasets (a 9600 images dataset for the tiles identification and a dataset composed by 2100 positive and 2430 negatives samples) and two test datasets - *small* and *big* - were built.

The *small* test dataset is composed by 288 images and was used to validate if the algorithm was able to recognise every single tile and also, to compare the performance of HOG and SVM with other object recognition methods in the domains of features extraction (Template Matching) and features detection and description (SURF, SIFT, ORB and BRISK). The tests executed on this dataset resulted in HOG and SVM outperforming the other methods with a classification accuracy of **100%**. SURF was the best of its kind with a 98.89% accuracy and, with an accuracy of 81.54%, *CCOEFF* was the best Template Matching version.

The *big* test dataset is composed by 12000 images and was used to exhaustively test the developed computer vision system, presenting a **99.99%** accuracy, being that only **12** out of the **96000** tiles, that make up this dataset, were wrongly classified.

All the objectives were met and the Tactode programming system, after this work, can rely on a rotation-invariant vision-based algorithm that achieve a **100%** and **99.99%** accuracy on the *small* and *big* datasets, respectively.

6.1 Contributions

The contributions of this dissertation are the following:

- New 3D-printed Tactode tiles;

- Computer vision system capable of detecting and recognising the Tactode tiles without ArUco markers;
- Two training datasets available online with Tactode tiles without ArUco markers;
- Two test datasets available online with Tactode codes composed by ArUco-less tiles for the Tactode programming system;
- Comparative study on different classification methods like HOG and SVM, SIFT, SURF, ORB, BRISK and Template Matching for identifying blocks with different shapes and inner symbols.

6.2 Future Work

The integration of the vision-based algorithm developed in this dissertation with the Tactode programming system is the main area to be exploit in the future.

In addition, using fiducial markers to help segmenting an image, obtaining the exact location of the inner contents of the tiles, would be an interesting approach to avoid errors such as the tiles borders being broken, compromising the remaining steps since the contours would not be well computed.

Lastly, a deep learning strategy could be also considered because it is a very current topic and an under development domain, and it would be interesting to compare its performance with the machine learning technique already implemented (HOG and SVM) in the vision-based system.

Appendix A

Datasets Test Codes

A.1 Tactode Codes of the *Small* Dataset

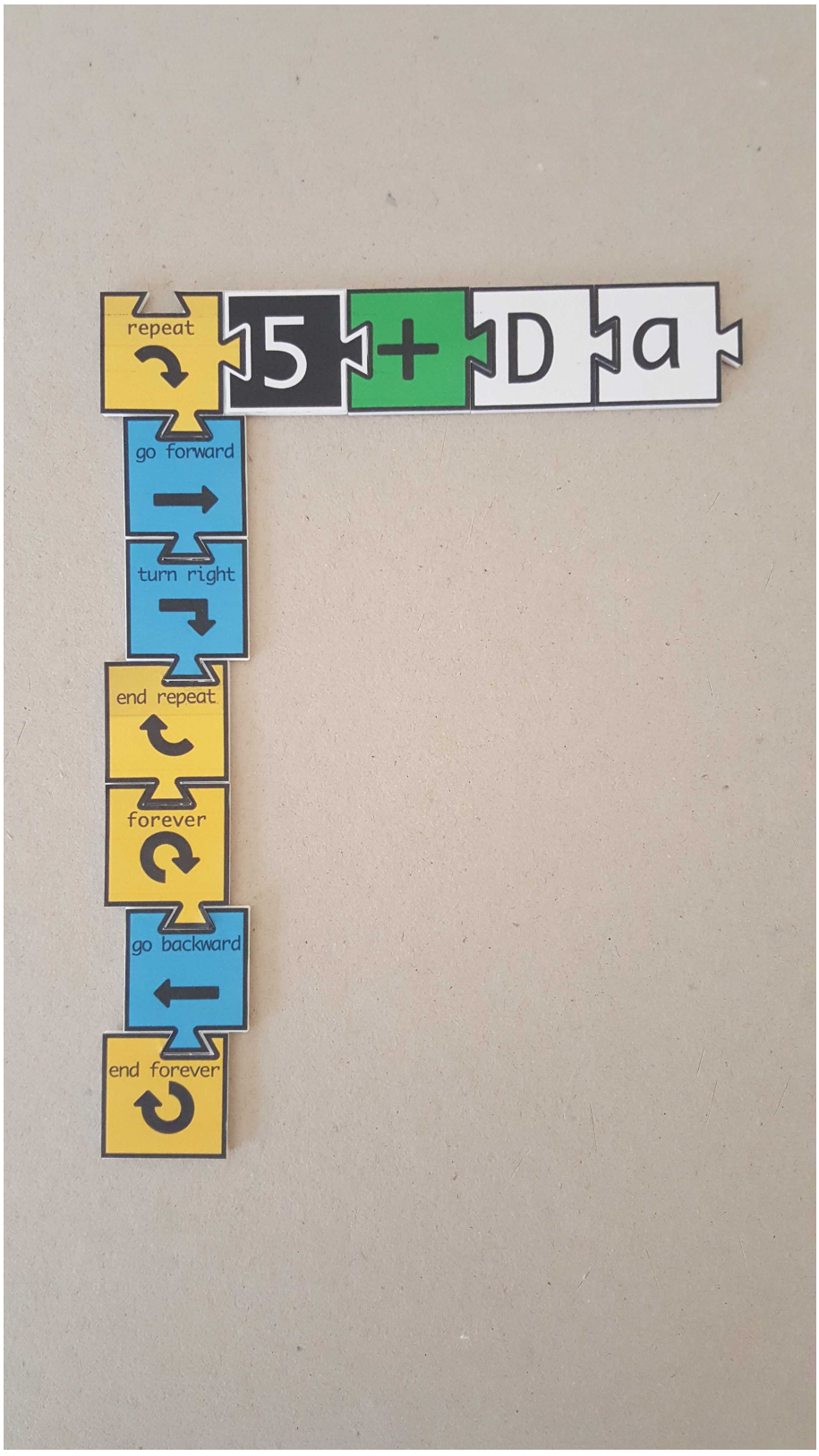


Figure A.1: Code with 11 tiles of the *small* dataset.

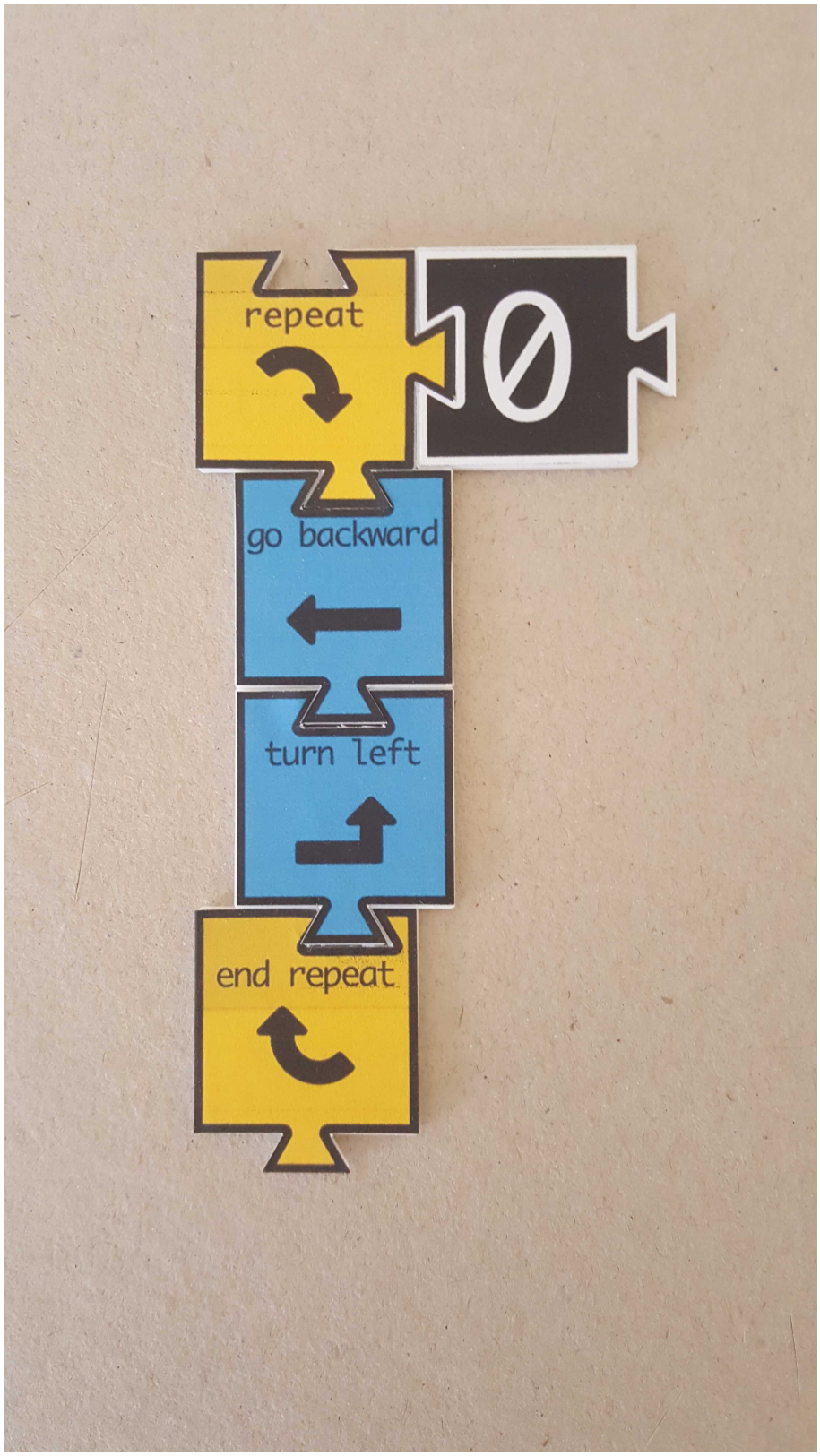


Figure A.2: Code with five tiles of the *small* dataset.

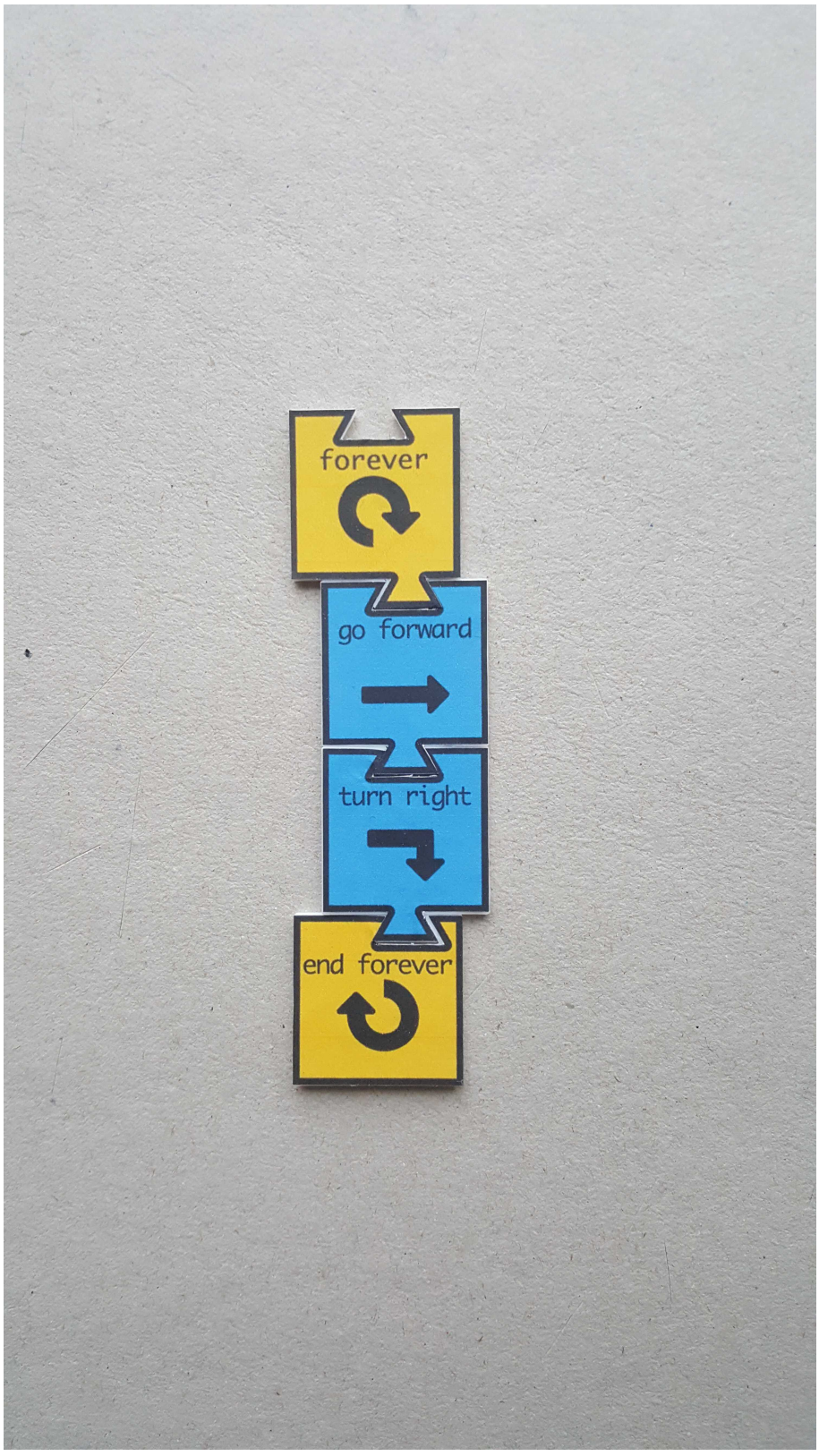


Figure A.3: Code with four tiles of the *small* dataset.

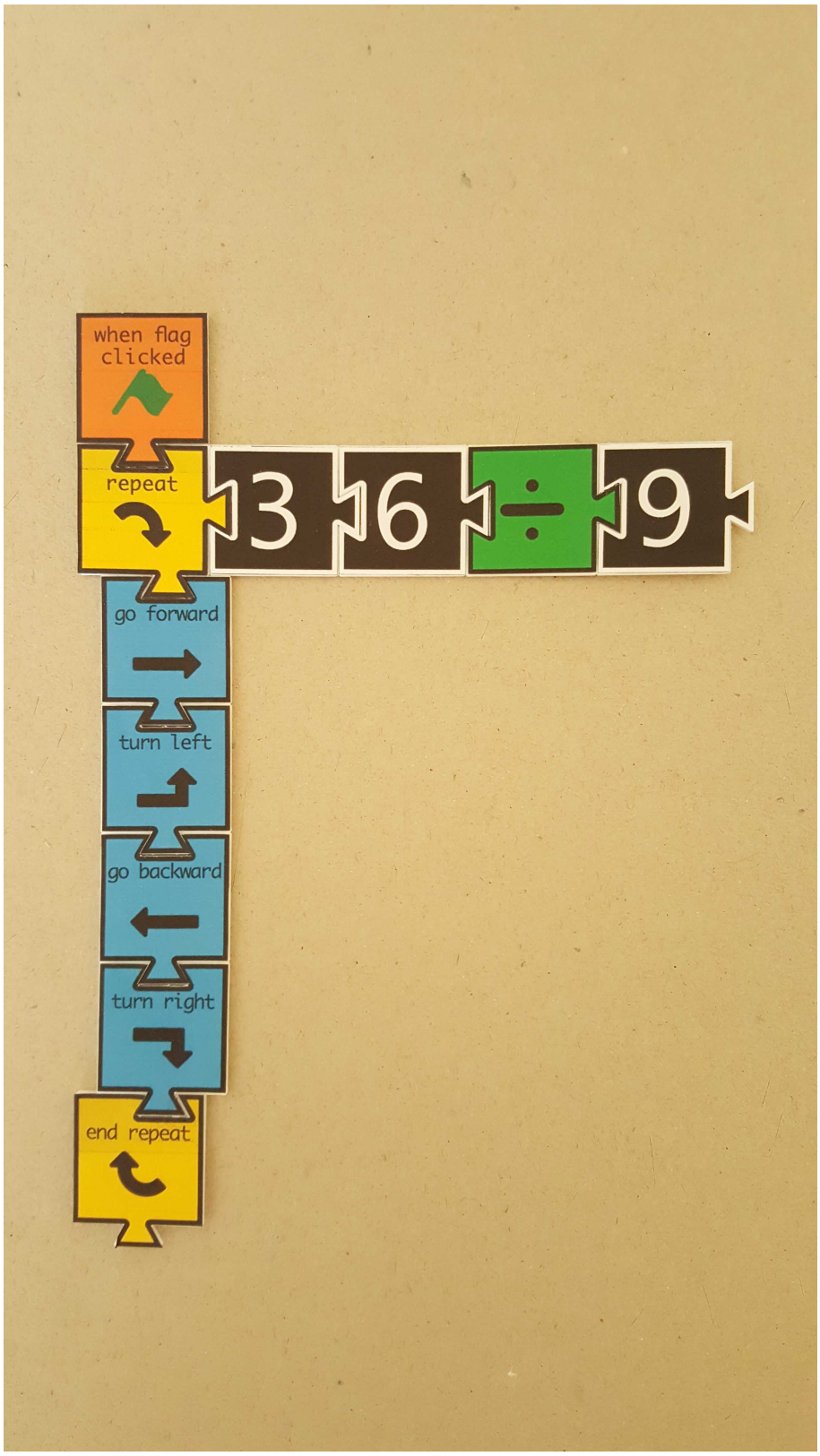


Figure A.4: Code with 11 tiles of the *small* dataset.

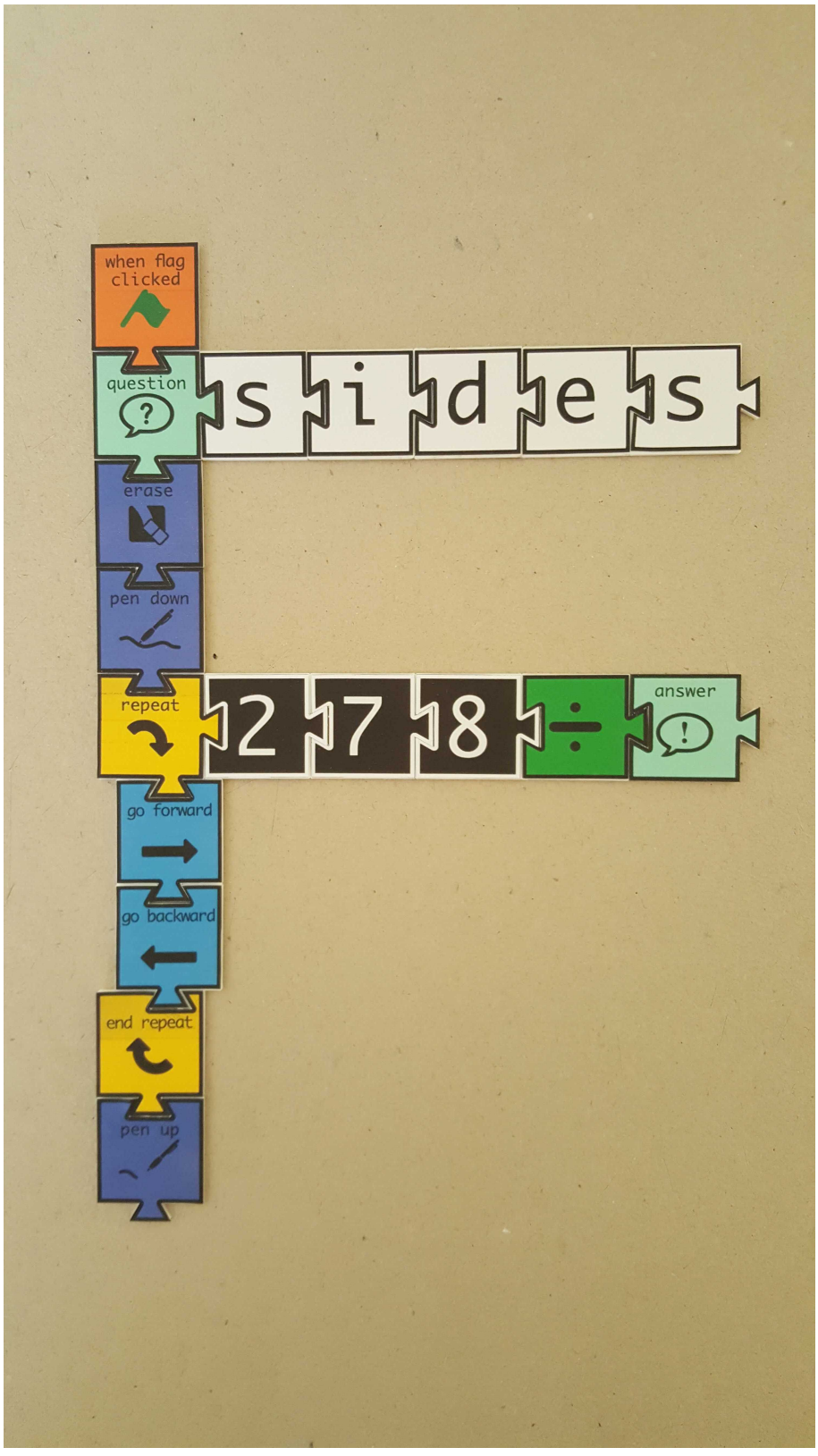


Figure A.5: Code with 19 tiles of the *small* dataset.

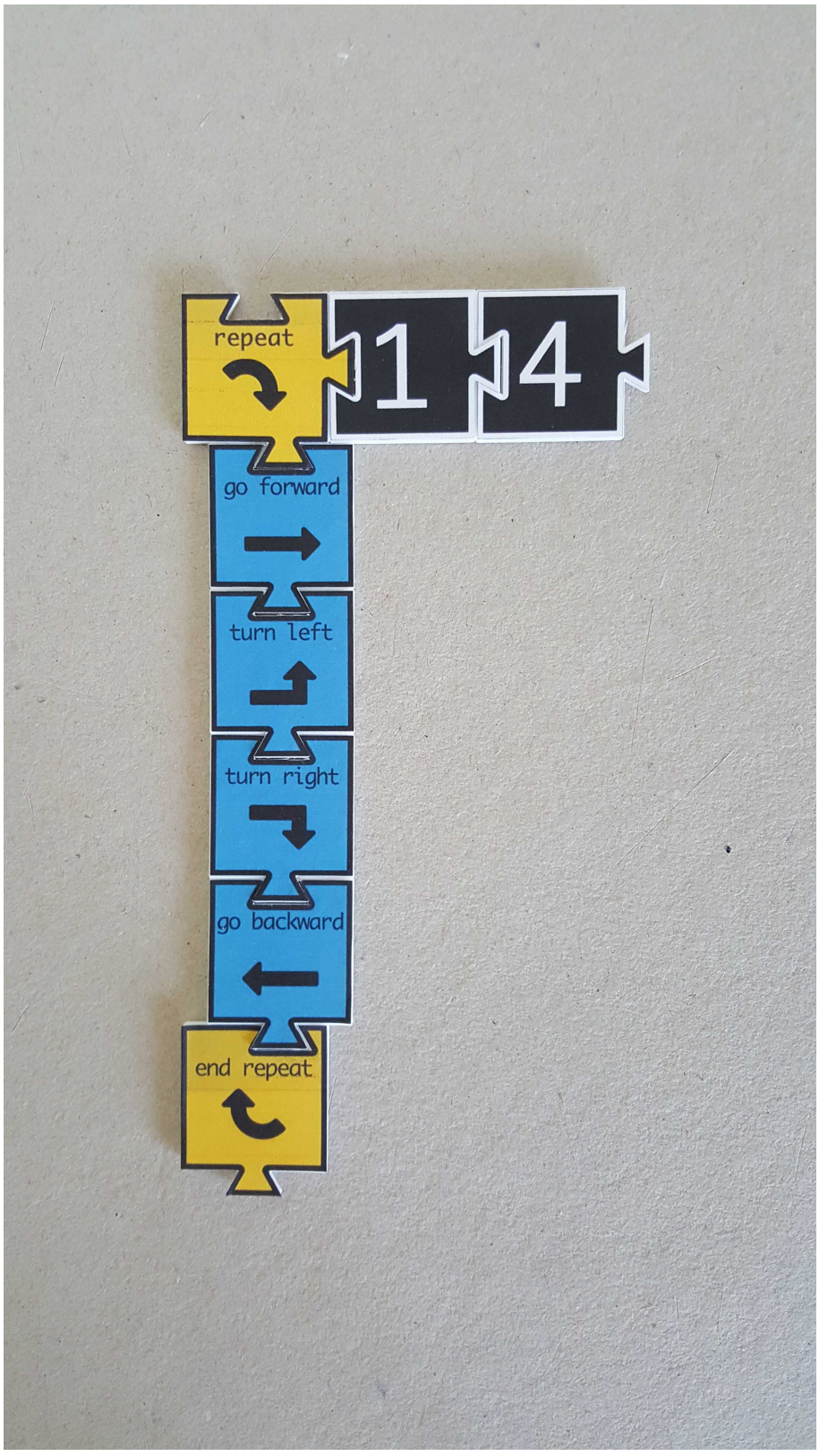


Figure A.6: Code with 8 tiles of the *small* dataset.

A.2 Tactode Codes of the *Big* Dataset

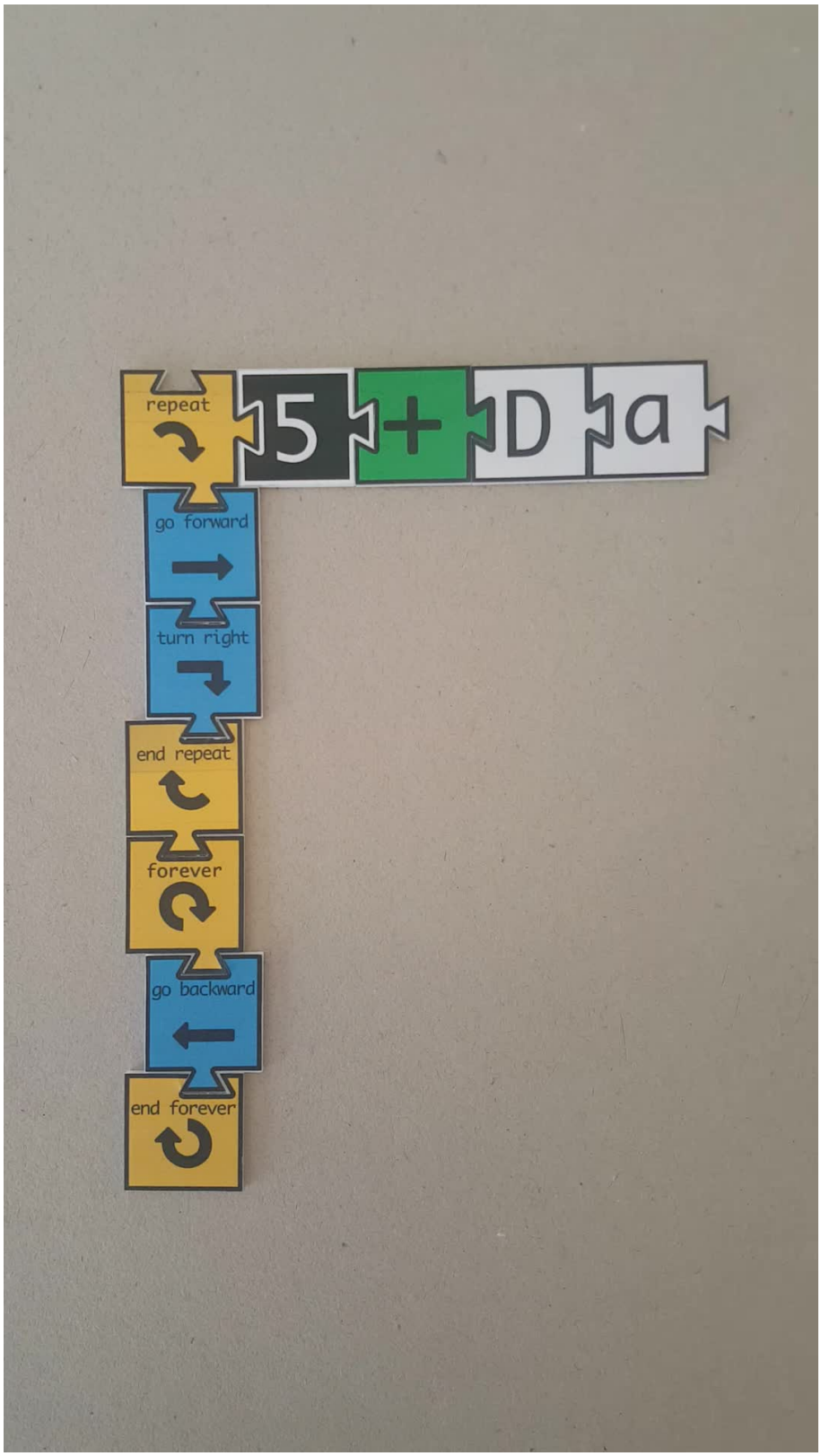


Figure A.7: Code with 11 tiles of the *big* dataset.

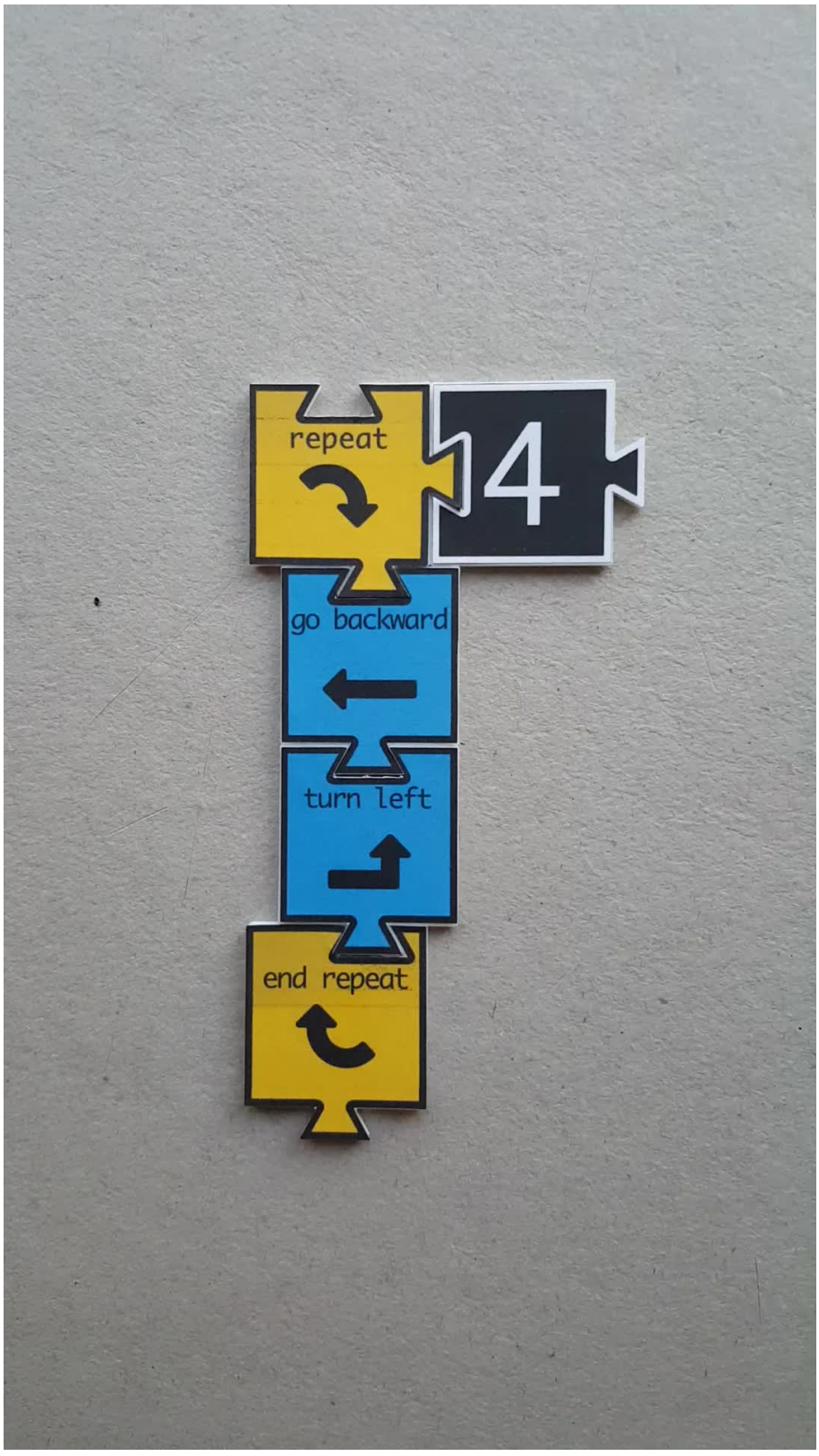


Figure A.8: Code with five tiles of the *big* dataset.

Appendix B

Confusion Matrices of the Tiles

B.1 Confusion Matrices Resulting from the *Small* Test Dataset

HOG - small		TRUTH																															
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
	end repeat	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240
	forever	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
	repeat	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240
	flag	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
	lower a	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	lower d	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	lower e	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	lower i	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	lower s	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
	upper D	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	backward	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240
	forward	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240
	left	0	0	0	0	0	0	0	0	0	0	0	0	0	144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	192
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	48
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	48
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	48
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	48
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	48
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	48
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	48
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	96
	answer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	48
	question	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	48
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	48
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	48
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	48
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	2784

Figure B.1: Confusion matrix of HOG and SVM resulting from the *small* dataset.

SURF		TRUTH																															
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97
	end repeat	0	240	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	243
	forever	0	0	96	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97
	repeat	0	0	0	240	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	243
	flag	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	97
	lower a	0	0	0	0	0	48	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49
	lower d	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47
	lower e	0	0	0	0	0	0	0	47	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49
	lower i	0	0	0	0	0	0	0	0	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
	lower s	0	0	0	0	0	0	0	0	0	84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84
	upper D	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	backward	0	0	0	0	0	0	0	0	0	1	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	246
	forward	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240
	left	0	0	0	0	0	0	0	0	0	1	0	0	0	144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	145
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	192	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	193
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	49
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	48
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	48
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	0	0	0	0	0	0	0	0	46
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	48
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	48
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	47	0	0	0	0	0	0	0	49
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	2	0	0	0	0	0	46
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	0	0	0	0	0	88
	answer	0	0	0	0	0	0	0	0	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	48	0	0	0	0	54
	question	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	48
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	48
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	48
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	48
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	2784

Figure B.2: Confusion matrix of SURF resulting from the *small* dataset.

SIFT		TRUTH																																
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	0	0	0	0	0	1	1	1	11	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	9	0	0	0	0	0	129
	end repeat	0	240	0	0	0	0	4	1	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	7	0	0	0	0	0	269
	forever	0	0	96	0	0	0	0	2	1	2	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	106
	repeat	0	0	0	240	0	1	1	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	9	2	0	0	0	0	0	259
	flag	0	0	0	0	95	1	0	1	5	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	1	0	113
	lower a	0	0	0	0	0	42	3	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53
	lower d	0	0	0	0	0	0	37	0	0	8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	47
	lower e	0	0	0	0	0	0	1	29	0	4	2	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	13	0	0	0	0	0	51
	lower i	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	23
	lower s	0	0	0	0	0	0	0	0	0	20	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	23
	upper D	0	0	0	0	2	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8
	backward	0	0	0	0	0	0	0	0	5	3	1	240	0	0	0	0	0	0	0	0	0	0	0	0	0	3	10	0	0	0	0	0	262
	forward	0	0	0	0	0	1	0	0	3	1	1	0	240	0	0	0	0	0	0	0	0	0	1	0	0	0	4	0	0	0	0	0	251
	left	0	0	0	0	0	0	0	1	1	0	0	0	0	144	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	148
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	192	0	0	0	0	0	0	0	0	0	0	2	4	0	0	0	0	0	198
	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	42	1	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	47
	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	47	0	0	0	0	0	2	3	0	0	3	0	0	0	0	0	58
	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	48	0	0	0	0	3	0	5	0	0	0	0	0	0	0	57
	3	0	0	0	0	0	0	0	1	0	5	0	0	0	0	0	0	0	0	48	0	0	0	0	3	9	2	0	0	0	0	0	0	68
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	48	0	0	0	0	3	0	1	0	0	0	0	0	53
	5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	48	1	0	0	0	1	0	0	0	0	0	0	51
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	44	0	0	3	0	1	0	0	0	0	0	49
	7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	40	2	0	0	0	0	0	0	0	0	43
	8	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	2	0	0	0	0	0	1	0	35	0	0	0	0	0	0	0	0	41
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	27
	addition	0	0	0	0	0	0	0	0	1	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	3	0	0	0	0	0	20
	division	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	13
	answer	0	0	0	0	1	0	1	6	6	10	12	0	0	0	0	0	0	0	0	0	0	1	0	1	0	7	15	48	0	0	0	0	108
	question	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	48	0	0	0	51
	erase	0	0	0	0	0	1	0	6	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	2	0	0	47	0	0	62
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	0	44
	pen erase	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	48	52
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	48	2784

Figure B.3: Confusion matrix of SIFT resulting from the *small* dataset.

BRISK		TRUTH																															
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	0	0	0	0	0	0	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	103
	end repeat	0	240	0	0	0	0	1	0	3	4	3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	254
	forever	0	0	96	0	0	0	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	102
	repeat	0	0	0	240	0	0	0	0	2	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	245
	flag	0	0	0	0	96	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98
	lower a	0	0	0	0	0	46	2	0	1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53
	lower d	0	0	0	0	0	1	45	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47
	lower e	0	0	0	0	0	0	0	48	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51
	lower i	0	0	0	0	0	0	0	0	13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14
	lower s	0	0	0	0	0	0	0	0	0	51	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53
	upper D	0	0	0	0	0	0	0	0	0	1	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	34
	backward	0	0	0	0	0	0	0	0	1	3	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	250
	forward	0	0	0	0	0	0	0	0	1	0	1	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	242
	left	0	0	0	0	0	0	0	0	2	2	0	0	0	144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	148
	right	0	0	0	0	0	0	0	0	1	1	1	0	0	0	192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	195
	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	46	3	2	0	0	0	0	4	0	0	2	0	0	0	0	0	60
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	39	1	0	0	1	0	2	0	1	0	0	0	0	0	0	45
	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	37	2	0	0	0	2	0	0	1	0	0	0	0	0	43
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	46	0	0	0	2	0	1	1	1	0	0	0	0	54
	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	2	0	48	0	0	4	0	0	0	1	0	0	0	0	58
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	44	1	1	0	0	0	0	0	0	0	0	47
	6	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	47	0	0	0	1	1	0	0	0	0	52
	7	0	0	0	0	0	0	0	0	5	5	2	0	0	0	0	0	1	1	0	0	0	0	30	0	0	1	7	0	0	0	0	52
	8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	50
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	46	0	0	0	0	0	0	49
	addition	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	2	0	0	0	0	45
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	60
	answer	0	0	0	0	0	0	0	0	6	12	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	6	48	0	0	0	74
	question	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	48	0	0	53
	erase	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	48	0	51
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	48	51
	pen erase	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	51
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	2784

Figure B.4: Confusion matrix of BRISK resulting from the *small* dataset.

ORB		TRUTH																																
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	0	0	0	0	0	102
	end repeat	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	243
	forever	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	99
	repeat	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	241
	flag	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	98
	lower a	0	0	0	0	0	47	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	52
	lower d	0	0	0	0	0	1	45	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	53
	lower e	0	0	0	0	0	0	0	48	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	53
	lower i	0	0	0	0	0	0	0	0	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
	lower s	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	99
	upper D	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	46
	backward	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	242
	forward	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	241
	left	0	0	0	0	0	0	0	0	1	0	0	0	0	144	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	146
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	192	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	193
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	49
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	49
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	48	0	0	0	0	0	0	0	2	0	0	0	0	0	51
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	48	1	0	0	0	0	0	1	0	0	0	0	0	51
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	2	0	0	0	0	0	0	0	0	0	48
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	1	0	0	1	1	0	0	0	0	0	51
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	1	0	0	0	0	0	0	46
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	48
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	48	0	0	0	0	0	0	0	49
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	36
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	answer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	48	0	0	0	88
	question	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	0	48	0	0	0	75
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	48	0	0	49
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	47
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	48
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	2784

Figure B.5: Confusion matrix of ORB resulting from the *small* dataset.

TM_CCOEFF		TRUTH																																
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	96	5	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117
	end repeat	0	194	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	194
	forever	0	0	80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80
	repeat	0	5	0	240	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	247
	flag	0	0	0	0	73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	73
	lower a	0	0	0	0	8	35	0	0	0	0	0	22	19	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	92
	lower d	0	0	0	0	13	11	48	0	0	0	0	8	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	1	0	91
	lower e	0	0	0	0	0	0	0	47	0	5	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	2	0	25	5	0	88
	lower i	0	36	0	0	0	0	0	0	48	10	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39	36	0	13	0	186
	lower s	0	0	0	0	0	0	0	0	0	81	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	85
	upper D	0	0	0	0	0	2	0	1	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51
	backward	0	0	0	0	0	0	0	0	0	0	0	205	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	38	295
	forward	0	0	0	0	0	0	0	0	0	0	0	5	187	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	198
	left	0	0	0	0	0	0	0	0	0	0	0	0	0	133	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	188	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	188
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	49
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	53
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	19
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	45
	5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	29	0	48	0	0	0	0	0	0	0	0	1	0	0	79
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	1	0	0	0	0	0	1	0	0	49
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	48	0	0	0	0	0	0	0	0	0	50
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	47
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	48	0	0	0	0	0	0	0	51
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37	35	0	0	0	0	0	72
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	61	0	0	0	0	0	72
	answer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	10
	question	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	2	15
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	9
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	2784

Figure B.6: Confusion matrix of Template Matching with *CCOEFF* resulting from the *small* dataset.

TM_SQDIFF		TRUTH																																
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	95	27	15	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	148
	end repeat	0	171	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	171
	forever	0	0	76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76
	repeat	0	32	1	224	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	263
	flag	0	3	0	0	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81
	lower a	1	0	1	2	1	48	36	4	0	10	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	106
	lower d	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12
	lower e	0	0	3	0	0	0	0	44	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	lower i	0	7	0	10	0	0	0	0	48	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	32	0	0	0	128
	lower s	0	0	0	4	0	0	0	0	0	74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	79
	upper D	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
	backward	0	0	0	0	0	0	0	0	0	0	0	118	33	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	154
	forward	0	0	0	0	0	0	0	0	0	0	0	36	123	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	178
	left	0	0	0	0	0	0	0	0	0	0	0	32	21	83	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	15	16	212
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	124
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	50
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	10	0	0	0	0	0	0	0	0	0	55
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	0	0	1	0	0	0	0	0	0	0	0	47
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	45
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	46	0	0	0	0	0	0	0	0	0	0	0	47
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	0	0	0	0	0	0	0	0	46
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	0	0	0	0	0	0	0	38
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	47	0	0	0	0	0	0	0	0	51
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	0	0	0	0	0	0	0	41
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	66	0	0	0	0	0	95
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	1	0	0	0	0	31
	answer	0	0	0	0	0	0	0	0	0	6	0	31	16	36	29	3	0	0	0	1	2	0	0	0	3	10	0	14	3	23	25	0	202
	question	0	0	0	0	0	0	0	0	0	3	0	23	47	5	13	0	0	0	0	0	0	0	0	0	0	9	0	4	12	1	1	0	118
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	5
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	5	32	39
	Total	96	240	96	240	96	48	48	48	48	96	48	240	240	144	192	48	48	48	48	48	48	48	48	48	48	48	96	48	48	48	48	48	2784

Figure B.7: Confusion matrix of Template Matching with *SQDIFF* resulting from the *small* dataset.

B.2 Confusion Matrices Resulting from the *Big* Test Dataset

HOG - big		TRUTH																																
		end forever	end repeat	forever	repeat	flag	lower a	lower d	lower e	lower i	lower s	upper D	backward	forward	left	right	0	1	2	3	4	5	6	7	8	9	addition	division	answer	question	erase	pen down	pen up	Total
PREDICTED	end forever	6000	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6010
	end repeat	0	11990	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11990
	forever	0	0	5998	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5998
	repeat	0	0	0	12000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12000
	flag	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	lower a	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	lower d	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	lower e	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	lower i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	lower s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	upper D	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	backward	0	0	0	0	0	0	0	0	0	0	0	12000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12000
	forward	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	left	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	6000
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	0	0	0	0	0	6000
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	addition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6000	0	0	0	0	0	0	6000
	division	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	answer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	question	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	pen down	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	pen erase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Total	6000	12000	6000	12000	0	6000	0	0	0	0	6000	12000	6000	6000	6000	0	0	0	0	6000	6000	0	0	0	0	6000	0	0	0	0	0	0	96000

Figure B.8: Confusion matrix of HOG and SVM resulting from the *big* dataset.

References

[1] Ângela Cardoso. Tangible language for educational programming of robots and other targets. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2019.

[2] Márcia Alves. Robotic simulator for the tactode tangible block programming system. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2019.

[3] Armando Sousa. Tactode. Available at https://fe.up.pt/asousa/tactode. Last accessed 2020-07-05.

[4] Márcia Alves, Armando Sousa, and Ângela Cardoso. Web based robotic simulator for tactode tangible block programming system. In *Robot 2019: Fourth Iberian Robotics Conference*, pages 490–501, 2020.

[5] Ionic. Ionic Framework. Available at https://ionicframework.com. Last accessed 2020-06-08.

[6] H.D. Cheng, X.H. Jiang, Y. Sun, and Jingli Wang. Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259 – 2281, 2001. doi:https://doi.org/10.1016/S0031-3203(00)00149-7.

[7] Noor Ibraheem, Mokhtar Hasan, Rafiqul Zaman Khan, and Pramod Mishra. Understanding color models: A review. *ARPN Journal of Science and Technology*, 2:265–275, 2012.

[8] Paul Hough. A method and means for recognition complex patterns; US Patent: US3069654A, 1962. Available at https://patents.google.com/patent/US3069654A/en. Last accessed 2020-06-10.

[9] O.D. Trier, A.K. Jain, and T Taxt. Feature extraction methods for character recognition - A survey. *Pattern Recognition*, 29(4):641–662, 1996.

[10] Sneha Jain and Vijaya Laxmi. Color image segmentation techniques: A survey. In *Proceedings of the International Conference on Microelectronics, Computing & Communication Systems*, pages 189–197, 2018.

[11] António Anjos and Hamid Shahbazkia. Bi-level image thresholding - a fast method. *BIOSIGNALS*, 2:70–76, 01 2008.

[12] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979. doi:10.1109/TSMC.1979.4310076.

[13] Asad Mohammed Khan. Image segmentation methods : A comparative study. *International Journal of Soft Computing and Engineering (IJSCE)*, 3:84–92, 2013.

[14] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. *In International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation*, 132, 1979.

[15] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. *Proceedings - 3rd Asia Pacific Computer Human Interaction, APCHI 1998*, pages 63–68, 1998. doi:10.1109/APCHI.1998.704151.

[16] M. Fiala. Comparing ARTag and ARToolkit plus fiducial marker systems. *HAVE 2005: IEEE International Workshop on Haptic Audio Visual Environments and their Applications*, 2005:148–153, 2005. doi:10.1109/HAVE.2005.1545669.

[17] J. Rekimoto and Y. Ayatsuka. Cybercode: Designing augmented reality environments with visual tags. *Proceedings of DARE 2000 on Designing Augmented Reality Environments*, pages 1–10, 2000. doi:10.1145/354666.354667.

[18] M. Fiala. Designing highly reliable fiducial markers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1317–1324, 2010. doi:10.1109/TPAMI.2009.146.

[19] Michael Rohs and Beat Gfeller. Using camera-equipped mobile phones for interacting with real-world objects. *Advances in Pervasive Computing*, 2004.

[20] Daniel Wagner and Dieter Schmalstieg. ARToolKitPlus for pose tracking on mobile devices. *Computer Vision Winter Workshop*, 2007.

[21] Daniel Flohr and Jan Fischer. A lightweight id-based extension for marker tracking systems. *Eurographics Symposium on Virtual Environments*, 2007.

[22] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. doi:10.1016/j.patcog.2014.01.005.

[23] Arjen Rhijn and Jurriaan Mulder. Optical tracking using line pencil fiducials. *Eurographics Symposium on Virtual Environments*, 2004. doi:10.2312/EGVE/EGVE04/035-044.

[24] F. Bergamasco, A. Albarelli, and A. Torsello. Image-space marker detection and recognition using projective invariants. *Proceedings - 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2011*, pages 381–388, 2011. doi:10.1109/3DIMPVT.2011.55.

[25] B. Atcheson, F. Heide, and W. Heidrich. CALTag: High precision fiducial markers for camera calibration. *VMV 2010 - Vision, Modeling and Visualization*, pages 41–48, 2010. doi:10.2312/PE/VMV/VMV10/041-048.

[26] E. Olson. AprilTag: A robust and flexible visual fiducial system. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011. doi:10.1109/ICRA.2011.5979561.

[27] J. Degol, T. Bretl, and D. Hoiem. ChromaTag: A colored marker and fast detection algorithm. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:1481–1490, 2017. doi:10.1109/ICCV.2017.164.

[28] F. Bergamasco, A. Albarelli, and A. Torsello. Pi-Tag: A fast image-space marker design based on projective invariants. *Machine Vision and Applications*, 24(6):1295–1310, 2013. doi:10.1007/s00138-012-0469-6.

[29] F. Bergamasco, A. Albarelli, E. Rodolà, and A. Torsello. RUNE-Tag: A high accuracy fiducial marker with strong occlusion resilience. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 113–120, 2011. doi:10.1109/CVPR.2011.5995544.

[30] L. Calvet, P. Gurdjos, and V. Charvillat. Camera tracking using concentric circle markers: Paradigms and algorithms. *Proceedings - International Conference on Image Processing, ICIP*, pages 1361–1364, 2012. doi:10.1109/ICIP.2012.6467121.

[31] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini. Detection and accurate localization of circular fiducials under highly challenging conditions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:562–570, 2016. doi:10.1109/CVPR.2016.67.

[32] Aldebaran Robotics. NAOqi. Available at http://doc.aldebaran.com/2-5/naoqi/vision/allandmarkdetection.html. Last accessed 2020-06-08.

[33] OpenCV. OpenCV. Available at https://docs.opencv.org/4.4.0/index.html. Last accessed 2020-07-04.

[34] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, 2:1150–1157, 1999.

[35] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. doi:10.1023/B:VISI.0000029664.99615.94.

[36] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. doi:10.1016/j.cviu.2007.09.014.

[37] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, Dec 2001. doi:10.1109/CVPR.2001.990517.

[38] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3951 LNCS:430–443, 2006. doi:10.1007/11744023_34.

[39] E. Mair, G.D. Hager, D. Burschka, M. Suppa, and G. Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6312 LNCS(PART 2):183–196, 2010. doi:10.1007/978-3-642-15552-9_14.

[40] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6314 LNCS(PART 4):778–792, 2010. doi:10.1007/978-3-642-15561-1_56.

[41] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2548–2555, 2011. doi:10.1109/ICCV.2011.6126542.

[42] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to sift or surf. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

[43] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 510–517, 2012. doi:10.1109/CVPR.2012.6247715.

[44] P.F. Alcantarilla, A. Bartoli, and A.J. Davison. KAZE features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7577 LNCS(PART 6):214–227, 2012. doi:10.1007/978-3-642-33783-3_16.

[45] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990. doi:10.1109/34.56205.

[46] P.F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *BMVC 2013 - Electronic Proceedings of the British Machine Vision Conference 2013*, 2013. doi:10.5244/C.27.13.

[47] X. Yang and K.-T. Cheng. LDB: An ultra-fast feature for scalable augmented reality on mobile devices. *ISMAR 2012 - 11th IEEE International Symposium on Mixed and Augmented Reality 2012, Science and Technology Papers*, pages 49–57, 2012. doi:10.1109/ISMAR.2012.6402537.

[48] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, I:886–893, 2005. doi:10.1109/CVPR.2005.177.

[49] S. A. K. Tareen and Z. Saleem. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, March 2018. doi:10.1109/ICOMET.2018.8346440.

[50] Ayon Dey. Machine Learning Algorithms: A Review. *International Journal of Computer Science and Information Technologies*, 7(3):1174–1179, 2016.

[51] Nabil H. Farhat and Vladimir Vapnik. Support-Vector Networks. *IEEE Expert-Intelligent Systems and their Applications*, 7(5):63–72, 1992. doi:10.1109/64.163674.

[52] Huaqing Li, Feihu Qi, and Shaoyu Wang. A comparison of model selection methods for multi-class support vector machines. *Lecture Notes in Computer Science*, 3483(IV):1140–1148, 2005.

[53] Ian G Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep learning. *Nature*, 521:436–444, 2015. doi:10.1038/nature14539.

[54] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. doi:10.1109/CVPR.2016.91.

[55] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 6517–6525, 2017. doi:10.1109/CVPR.2017.690.

[56] Joseph Redmon and Ali Farhadi. YOLO v.3. Technical report, 2018. Available at https://pjreddie.com/media/files/papers/YOLOv3.pdf. Last accessed 2020-06-09.

[57] David Powers and Ailab. Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation. *J. Mach. Learn. Technol*, 2:2229–3981, 2011. doi:10.9735/2229-3981.

[58] OpenSCAD. OpenSCAD. Available at https://www.openscad.org/index.html. Last accessed 2020-06-08.

[59] Ultimaker. Ultimaker Cura. Available at https://ultimaker.com/software/ultimaker-cura. Last accessed 2020-06-08.

[60] Mozilla. JavaScript. Available at https://developer.mozilla.org/en-US/docs/Web/JavaScript. Last accessed 2020-06-11.

[61] Node.js. Node.js. Available at https://nodejs.org/en. Last accessed 2020-06-11.

[62] npm. npm. Available at https://www.npmjs.com. Last accessed 2020-06-11.

[63] Vincent Mühler. opencv4nodejs. Available at https://www.npmjs.com/package/opencv4nodejs. Last accessed 2020-06-20.

[64] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[65] Lily Rae. opencvjs-circles. Available at https://github.com/lilyrae/opencvjs-circles. Last accessed 2020-06-28.