Erasmus Mundus Joint Master Degree in Image Processing and Computer Vision (EMJMD-IPCV)

Applied Video Sequence Analysis
Lab 4 “Histogram-based object tracking”

Instructor: Juan Carlos San Miguel (Juancarlos.Sanmiguel@uam.es)

Teaching assistant: Paula Moral (Paula.Moral@uam.es)

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY

UAM
Universidad Autónoma de Madrid

université de BORDEAUX

OBJECTIVES

- **Develop one algorithm for single-object tracking based on histograms** using the OpenCV C++ library

- **Analyze the strengths and weaknesses** of the different modules/features that compose the pipeline of the algorithm

- **Increase independency and self-learning skills**, must implement algorithms directly from research papers.

DELIVERY RULES

- Assignment available on Moodle to submit your material
- The material must be submitted as a ZIP file with the following format *Surname1name1_ Surname2name2_lab4.zip*
- The submitted ZIP file will contain
  - Report in PDF format following the guidelines (max 12 pages)
  - For each task, a directory containing:
    - *Makefile* to compile and link the program by simply running *make (Suggestion: use the makefile provided for lab1)*
    - "src" directory with all source files (.h, .hpp, .c and .cpp) necessary for compiling and executing the corresponding program in Linux
    - **Please do not submit configuration files of Eclipse**

TASKS

Seven tasks

4.0 Code template for processing videos

4.1 Color-based tracking: code implementation
4.2 Color-based tracking: analysis over real data

4.3 Gradient-based tracking: code implementation
4.4 Gradient-based tracking: analysis over real data

4.5 Color&Gradient-based tracking: code implementation
4.6 Color&Gradient-based tracking: analysis over real data

4.0 Code template for processing videos

Download the code template and create an Eclipse CDT++ project

Use this template as baseline for all tasks

This template can be compiled and executed, allowing

- To iterate smoothly over all test video sequences for Lab4
- To save a video file with results of each sequence using the `VideoWriter` class
- To read groundtruth data for each video sequence by using the function (i.e. groundtruth as manual annotations of the object location as bounding boxes)

```
std::vector<cv::Rect> readGroundTruthFile(std::string groundtruth_path);
```

- To compare groundtruth data and estimated data (both as bounding boxes) for each video sequence by using the function

```
std::vector<float> estimateTrackingPerformance(std::vector<cv::Rect> Bbox_GT,
std::vector<cv::Rect> Bbox_est);
```

- The estimated tracking result must be stored in the variable `list_bbox_est`

Help tips:

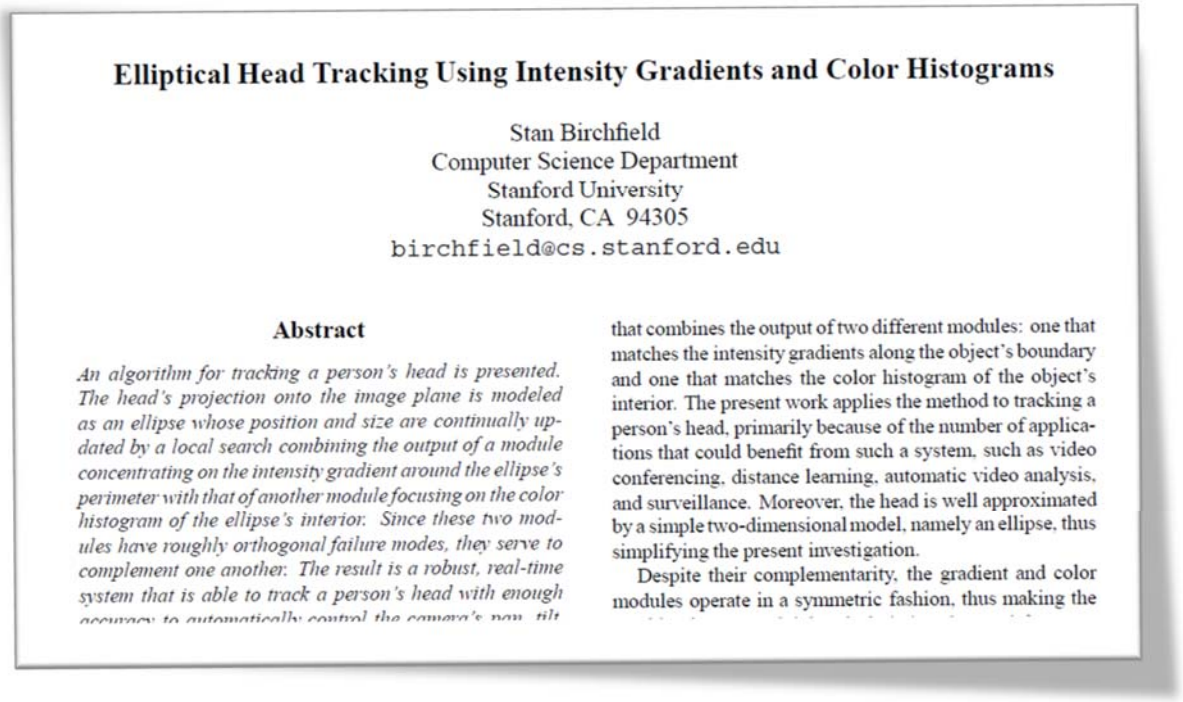
- The code uses the `std::vector` class to manage lists of items (floats, Rects,…). Please become familiar with its usage http://www.cplusplus.com/reference/vector/vector/vector/
- Functions are coded in the files `utils.cpp` and `utils.hpp`

TASKS

4.1 Color-based tracking: code implementation

4.2 Color-based tracking: analysis over real data

Objective: implement the color-based tracker described in the paper "Elliptical Head Tracking Using Intensity Gradients and Color Histograms"



Elliptical Head Tracking Using Intensity Gradients and Color Histograms

Stan Birchfield
Computer Science Department
Stanford University
Stanford, CA 94305
birchfield@cs.stanford.edu

Abstract

An algorithm for tracking a person's head is presented. The head's projection onto the image plane is modeled as an ellipse whose position and size are continually updated by a local search combining the output of a module concentrating on the intensity gradient around the ellipse's perimeter with that of another module focusing on the color histogram of the ellipse's interior. Since these two modules have roughly orthogonal failure modes, they serve to complement one another. The result is a robust, real-time system that is able to track a person's head with enough accuracy to automatically control the camera's pan, tilt

that combines the output of two different modules: one that matches the intensity gradients along the object's boundary and one that matches the color histogram of the object's interior. The present work applies the method to tracking a person's head, primarily because of the number of applications that could benefit from such a system, such as video conferencing, distance learning, automatic video analysis, and surveillance. Moreover, the head is well approximated by a simple two-dimensional model, namely an ellipse, thus simplifying the present investigation.

Despite their complementarity, the gradient and color modules operate in a symmetric fashion, thus making the

4.1 Color-based tracking: code implementation

4.2 Color-based tracking: analysis over real data

Use the code template as starting code for this task

Consider that your tracking algorithm must:

- Initialize an object model by using only the first frame and the groundtruth data
- Generate candidate locations using the grid approach described.
  (Focus only on object centers. Keep fixed the width/height of object location)
- Compare object and candidate models using the Battacharyya distance
- Select the candidate minimizing the Battacharyya distance (Eq1 w/o gradients)
  (please note that Eq1 maximizes similarity, here we will minimize distance)
- Your algorithm must be able to change the following parameters:
  number of histograms bins (`#bins`) and number of generated candidates (`#cand`)

Modifications of the original paper:

- As color features use gray-level from RGB, H channel (HSV), S channel (HSV), R channel (RGB), G channel (RGB) and B channel (RGB).
- Please do not implement the gradient feature and fusion approach in the paper

4.1 Color-based tracking: code implementation

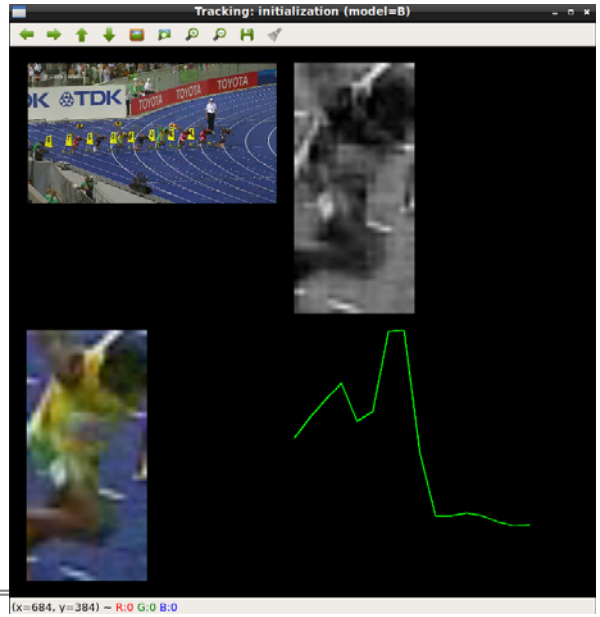
4.2 Color-based tracking: analysis over real data

Test sequences (1): “bolt1”

(You should get decent results for all features with #bins=16 and #cand>80 for frames 0-150)

Help tips

- Color conversion https://docs.opencv.org/3.4.4/de/d25/imgproc_color_conversions.html
- Histogram creation https://docs.opencv.org/3.4.4/d8/dbc/tutorial_histogram_calculation.html
- Histogram comparison https://docs.opencv.org/3.4.4/d8/dc8/tutorial_histogram_comparison.html
- Suggestions (optional):
  - Use `std::vector` class for accumulation https://bit.ly/395HDRY
  - Use `cv::putText` to plot text over images
  - Use the `ShowManyImages` class to display many images at once (e.g. current frame, candidate image and feature and histogram)
  - Avoid large portions of code in the main function
  - Use/create classes and methods whenever possible
  - Add comments to your code (at least functional units/methods)



TASKS

4.1 Color-based tracking: code implementation

4.2 Color-based tracking: analysis over real data

Objective: analyze and tune the Color-based tracker to get the max performance in real sequences from https://www.votchallenge.net/

Test sequences: "sphere", "car1"

1) Analyze the tracking problems that exist for test sequences

2) Starting from the code developed in task 4.1, assume 16 bins for computing the histograms and explore the effect in tracking performance (time and accuracy)…

a) …of the different color features implemented (e.g. Gray,H,S,R,G,B)

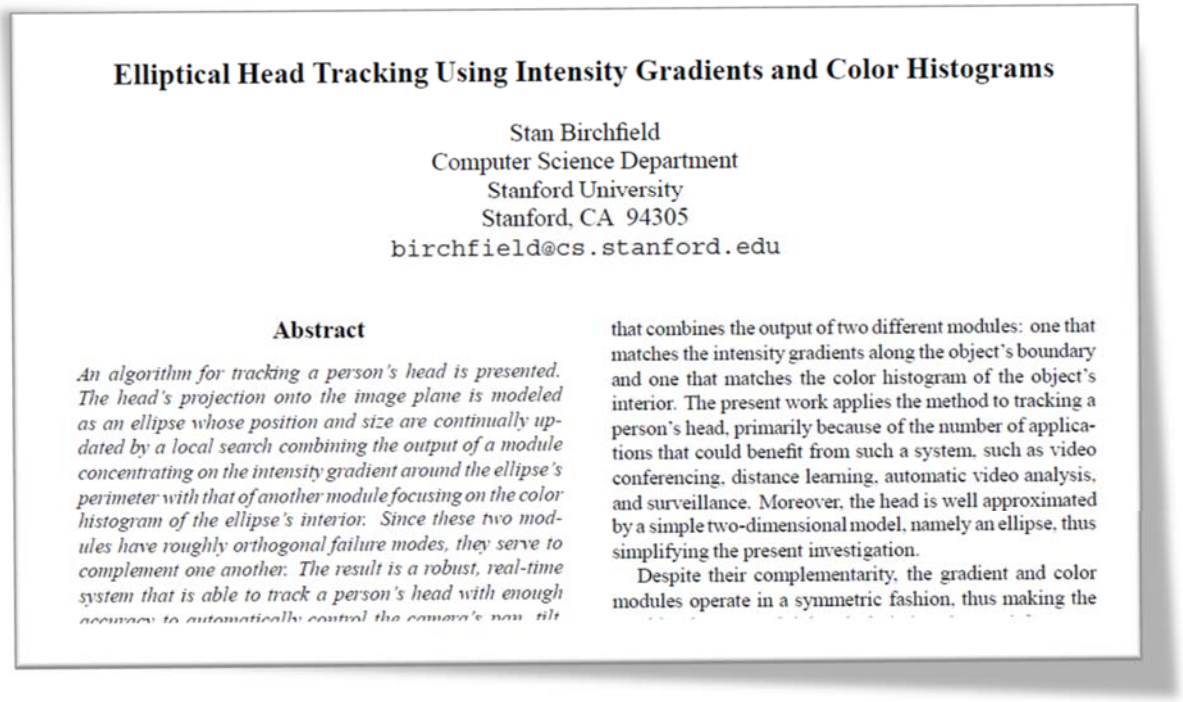
b) …of the changes in the parameter #cand (e.g. higher/lower than 100)

(To get the max grade, you do not need to test all possible variations for all sequences, provide a subset of reasonable experiments and solid conclusions over them)

4.3 Gradient-based tracking: code implementation

4.4 Gradient-based tracking: analysis over real data

Objective: implement the gradient-based tracker described in the paper "Elliptical Head Tracking Using Intensity Gradients and Color Histograms"



Elliptical Head Tracking Using Intensity Gradients and Color Histograms

Stan Birchfield
Computer Science Department
Stanford University
Stanford, CA 94305
birchfield@cs.stanford.edu

Abstract

An algorithm for tracking a person's head is presented. The head's projection onto the image plane is modeled as an ellipse whose position and size are continually updated by a local search combining the output of a module concentrating on the intensity gradient around the ellipse's perimeter with that of another module focusing on the color histogram of the ellipse's interior. Since these two modules have roughly orthogonal failure modes, they serve to complement one another. The result is a robust, real-time system that is able to track a person's head with enough accuracy to automatically control the camera's pan, tilt

that combines the output of two different modules: one that matches the intensity gradients along the object's boundary and one that matches the color histogram of the object's interior. The present work applies the method to tracking a person's head, primarily because of the number of applications that could benefit from such a system, such as video conferencing, distance learning, automatic video analysis, and surveillance. Moreover, the head is well approximated by a simple two-dimensional model, namely an ellipse, thus simplifying the present investigation.

Despite their complementarity, the gradient and color modules operate in a symmetric fashion, thus making the

Changes versus task 4.2 in red color

4.3 Gradient-based tracking: code implementation

4.4 Gradient-based tracking: analysis over real data

Use the code from task 4.1 as starting code for this task

Consider that your tracking algorithm must:

- Initialize an object model by using only the first frame and the groundtruth data
- Generate candidate locations using the grid approach described.
  (Focus only on object centers. Keep fixed the width/height of object location)
- Compare object and candidate models using the L2 distance.
- Select the candidate minimizing the L2 distance. (Eq 1 without the color score)
- Your algorithm must be able to change the following parameters:
  number of bins (`#bins`) and number of generated candidates (`#cand`)
  (`#bins` allows changing HOG descriptor length. Use default values for other parameters)

Modifications of the original paper:

- As gradient features, compute the HOG descriptor using the `HOGdescriptor` class of OpenCV (Do not code the gradient feature suggested in the paper)
- Please do not implement the fusion approach described in the paper

4.3 Gradient-based tracking: code implementation

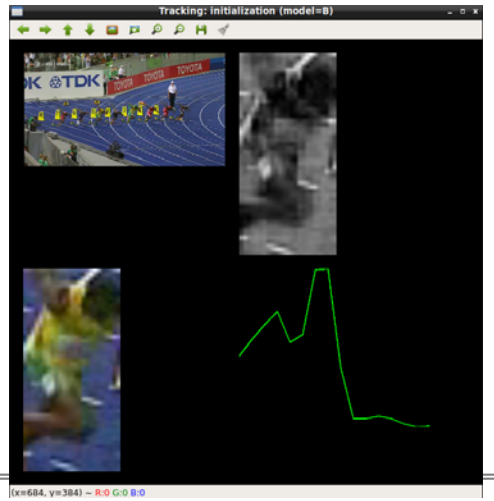
4.4 Gradient-based tracking: analysis over real data

Test sequences: "bolt1"

(You should get decent results for frames 0-150 with #bins=9 and #cand=100)

Help tips

- Color conversion https://docs.opencv.org/3.4.4/de/d25/imgproc_color_conversions.html
- HOG descriptor https://docs.opencv.org/3.4.4/d5/d33/structcv_1_1HOGDescriptor.html
- HOG computation example (lines 113-149 of the following code https://github.com/opencv/opencv/blob/master/samples/cpp/train_HOG.cpp
- Descriptor comparison https://docs.opencv.org/3.4.4/d8/dc8/tutorial_histogram_comparison.html
- Suggestions (optional):
  - Use `std::vector` class for accumulation https://bit.ly/395HDRY
  - Use `cv::putText` to plot text over images
  - Use the `ShowManyImages` class to display many images at once (e.g. current frame, candidate image, candidate features and HOG desc.)
  - Avoid large portions of code in the main function
  - Use/create classes and methods whenever possible
  - Add comments to your code (at least functional units/methods)



TASKS

4.3 Gradient-based tracking: code implementation

4.4 Gradient-based tracking: analysis over real data

Objective: analyze and tune the HOG-based tracker to get the max performance in real sequences from https://www.votchallenge.net/

Test sequences: "basketball","ball2"

1) Analyze the tracking problems that exist for test sequences

2) Starting from the code developed in task 4.3, explore the effect in tracking performance (visually and quantitatively)…

a) …of changes in the parameter `#bins` (e.g. higher/lower than 9)

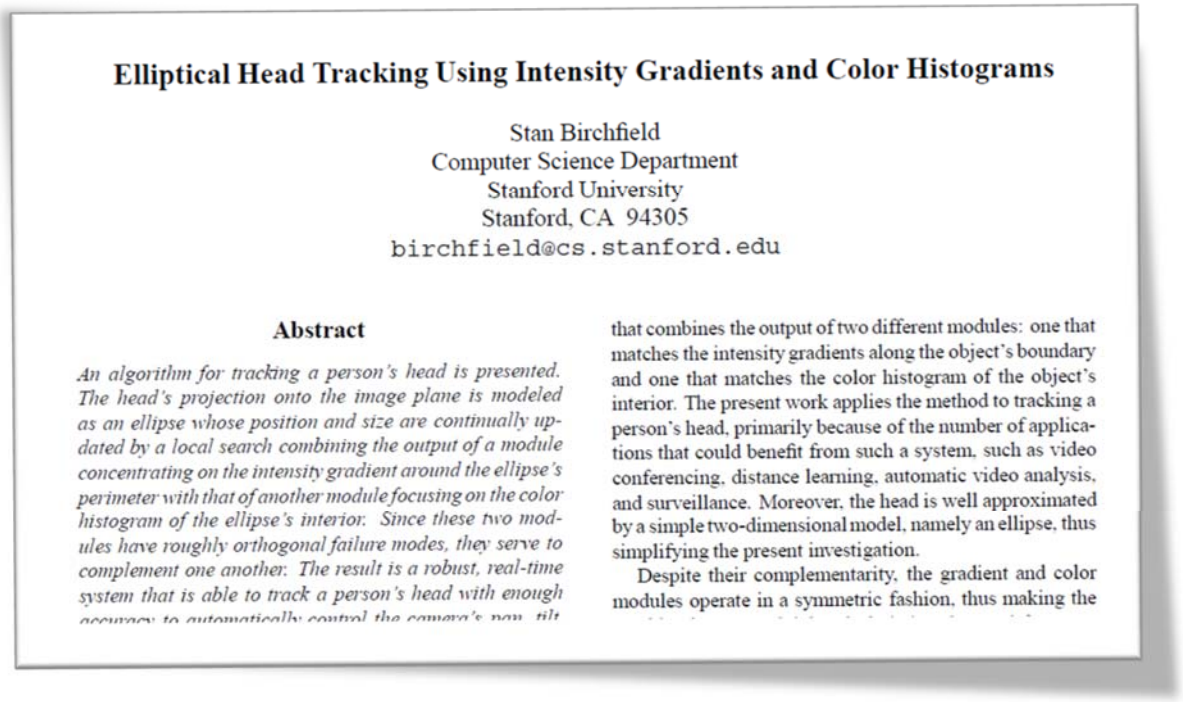
b) …of changes in the parameter `#cand` (e.g. higher/lower than 100)

(To get the max grade, you do not need to test all possible variations for all sequences, provide a subset of reasonable experiments and solid conclusions over them)

4.5 Color&Gradient-based tracking: code implementation

4.6 Color&Gradient-based tracking: analysis over real data

Objective: implement the fusion process of gradient and color information described in the paper "Elliptical Head Tracking Using Intensity Gradients and Color Histograms"



Elliptical Head Tracking Using Intensity Gradients and Color Histograms

Stan Birchfield
Computer Science Department
Stanford University
Stanford, CA 94305
birchfield@cs.stanford.edu

Abstract

An algorithm for tracking a person's head is presented. The head's projection onto the image plane is modeled as an ellipse whose position and size are continually updated by a local search combining the output of a module concentrating on the intensity gradient around the ellipse's perimeter with that of another module focusing on the color histogram of the ellipse's interior. Since these two modules have roughly orthogonal failure modes, they serve to complement one another. The result is a robust, real-time system that is able to track a person's head with enough accuracy to automatically control the camera's pan, tilt

that combines the output of two different modules: one that matches the intensity gradients along the object's boundary and one that matches the color histogram of the object's interior. The present work applies the method to tracking a person's head, primarily because of the number of applications that could benefit from such a system, such as video conferencing, distance learning, automatic video analysis, and surveillance. Moreover, the head is well approximated by a simple two-dimensional model, namely an ellipse, thus simplifying the present investigation.

Despite their complementarity, the gradient and color modules operate in a symmetric fashion, thus making the

4.5 Color&Gradient-based tracking: code implementation

4.6 Color&Gradient-based tracking: analysis over real data

Merge the code from task 4.1 & task 4.3 as starting code for this task

Consider that your tracking algorithm must:

- Initialize an object model by using only the first frame and the groundtruth data
- Generate candidate locations using the grid approach described.
  (Focus only on object centers. Keep fixed the width/height of object location)
- For each feature, obtain the scores by comparing object and candidate models. Then, obtain the normalized scores for each feature.
- Select the candidate minimizing the combination of both normalized scores
- Your algorithm must be able to choose each feature (color/gradient) or fusion.

Modifications of the original paper:

- As color features, use the color channel with highest performance in task 4.2
- As gradient features, use HOG from task 4.3
- As settings for both color and gradient features, use the number of candidates and descriptor length based on your conclusions for tasks 4.2 and 4.4

Test sequences: "bolt1" (You should get decent results for frames 0-150 with #bins=9 and #cand=100)

TASKS

4.5 Color&Gradient-based tracking: code implementation

4.6 Color&Gradient-based tracking: analysis over real data

Objective: analyze and tune the fusion-based tracker to get the max performance in real sequences from https://www.votchallenge.net/

Test sequences: "bag", "ball" and "road"

(as parameters, it is suggested to use #bins>=9 and #cand>=225)

1) Analyze the tracking problems that exist for test sequences

2) Starting from the code developed in task 4.5, compare the effect in tracking performance (visually and quantitatively) of using one feature (color or HOG) versus using the fusion approach

TASKS

- Expected workload (~20 hours total)

TASK	Expected hours	Type of work
Task 4.0	**1.5 (7.5%)**	**Read paper 1 and tasks of lab4** **Understand sample code**
Task 4.1	**6 (30%)** **1.5 (7.5%)**	**Source code** **Description of the code in the report**
Task 4.2	**2.5 (12.5%)**	**Experiments** **Description/analysis in the report**
Task 4.3	**2 (10%)** **0.5 (2.5%)**	**Source code** **Description of the code in the report**
Task 4.4	**2.5 (12.5%)**	**Experiments** **Description/analysis in the report**
Task 4.5	**0.5 (2.5%)** **0.5 (2.5%)**	**Source code** **Description of the code in the report**
Task 4.6	**2.5 (12.5%)**	**Experiments** **Description/analysis in the report**
TOTAL	**20 (100%)**	

EVALUATION (10 points)

Concept evaluated	TASK[1]	Max. Score	Criteria evaluated (described in the rubric available in Moodle)
Source Code (5 points)	**Task 4.1**	**3.5**	**Code: Functional requirements (60%)** **Code: Design & structure (30%)** **Code: Style (10%)**
	Task 4.3	**1**	**Code: Functional requirements (60%)** **Code: Design & structure (40%)**
	Task 4.5	**0.5**	**Code: Functional requirements (100%)**
Report[1] (5 points)	**Task 4.1**	**0.5**	**Report[2]: introduction & methods (100%)**
	Task 4.3	**0.25**	
	Task 4.5	**0.25**	
	Task 4.2	**1.5**	**Report[3]: Experimental methodology[3] (40%)** **Report[3]: Analysis & discussion (60%)**
	Task 4.4	**1.5**	
	Task 4.6	**1**	
	TOTAL	**10**	

[1]For the report, it is suggested to have and overall introduction and three main sections: section I (tasks 4.1 & 4.2), section II (tasks 4.3 and 4.4) and section III (tasks 4.5 and 4.6). Each section may have three subsections: methods, methodology and discussion.

[2]This criterion is applied together for all tasks, so only one grading item is defined in total

[3]These criteria are applied independently for each task, so 6 grading items are defined in total

EVALUATION (10 points)

Concept evaluated	TASK[1]	Max. Score	Criteria evaluated (described in the rubric available in Moodle)
Source Code (5 points)	**Task 4.1**	**3.5**	**Code: Functional requirements (60%)** **Code: Design & structure (30%)** **Code: Style (10%)**
	Task 4.3	**1**	**Code: Functional requirements (60%)** **Code: Design & structure (40%)**
	Task 4.5	**0.5**	**Code: Functional requirements (100%)**
Report[1] (5 points)	**Task 4.1**	**0.5**	**Report[2]: introduction & methods (100%)**
	Task 4.3	**0.25**	
	Task 4.5	**0.25**	
	Task 4.2	**1.5**	**Report[3]: Experimental methodology[3] (40%)** **Report[3]: Analysis & discussion (60%)**
	Task 4.4	**1.5**	
	Task 4.6	**1**	
	TOTAL	**10**	

Penalties:

- Delivery not following requirements: -0.5 points
- Late delivery after the remaining days of the late policy (remember 4-days in total for all labs)
  - -25% (one day), -50% (two days), -75% (three days), -100%(>= four days)

HELP

- Check OpenCV documentation for finding specific functions at https://docs.opencv.org/3.4.4/