М Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

АВТОМАТИЗАЦІЯ ПРЕЗЕНТАЦІЙ ЗА ДОПОМОГОЮ СКРАЙБІНГУ

Текстова частина до курсової роботи за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи
к.т.н. ст. викл. Афонін А. О.

_____
(підпис)
“____” _____ 2021 р.

Виконав студент Філоненко М. І.
“____” _____ 2021 р.

Київ 2021

Зміст

Анотація

У наш час електронні презентації стали однією з форм вираження думок. Головна їх мета – розкриття певної інформації, проте їх ефективність може вимірюватися не лише об’єктивними чинниками. Одна з частин презентації – правильне враження, яке вона має залишити, і не останню роль у цьому враженні відіграють анімації. Ця робота розкриває тему створення анімацій «відмальовки», тобто імітації поступового проявлення зображення так, як це відбувається під час рисування або малювання. Така анімація робить картинки динамічними та привертає увагу до презентації, що стимулює краще засвоєння матеріалу презентації.

Вступ

Актуальність теми. Розвиток електроніки та інформаційних технологій призвів до виникнення нових форм і засобів передачі інформації. Вони повністю замінили більшість форм віддаленої комунікації, якими раніше користувалося суспільство.

Серед форм передачі інформації однією з основних є форма доповіді або ж презентації, коли одна людина доносить свою думку відразу до багатьох. Однозначної відповіді на питання, який найбільш ефективний спосіб такої комунікації, наразі немає. Однак, електронні презентації, що набули відомості за рахунок популяризації програми Microsoft PowerPoint, стали вагомою частиною більшості доповідей, презентацій та виступів.

Microsoft PowerPoint та його аналоги справді надають потужні можливості для створення презентацій, проте їх використання наразі здебільшого обмежене найпримітивнішими застосуваннями. Це пояснюється відносною складністю як використання його можливостей під час розробки, так і під час демонстрації презентації. Основною ж проблемою рушія стало те, що без цих можливостей презентації часто стають надто статичними, що заважає виконанню основної цілі презентації – привернення уваги глядачів до інформації.

В цілому, існує декілька шляхів надання презентації динамічності; анімації є одним з таких засобів. Якщо раніше вони часто застосовувалися хаотично та без міри, сьогодні загальний рівень очікування користувачів від GUI такий, що анімації мають бути частиною презентації та органічно поєднувати її частини, а не бути просто цікавими ефектами. Зі збільшенням рівня очікувань має ускладнюватися і візуальна частина анімацій.

Одними із найбільш вживаних типів анімацій є анімація появи об’єктів. Якщо ми говоримо про презентації, що мають не просто красиво переміщувати об’єкти, а й пояснювати глядачеві їх виникнення, то для

текстових елементів найбільш доречна анімація друку, що легко реалізується за допомогою поступового відображення рядків.

Набагато більшою проблемою є відповідна анімація для зображень, тобто анімація їх малювання або рисування. На відміну від текстової інформації, зміст зображення, поданий у растровому форматі, не може бути розділений на окремі неподільні частини, які можна відображати одна за одною. Для цього доводиться застосовувати деякі евристики, такі як сегментація зображення та алгоритми нарощування регіонів (region growing).

У випадку створення нових анімацій для презентацій, вагомою проблемою є не лише розробка відповідних генераторів, але й донесення до кінцевого користувача. Наразі на ринку презентацій існують три вагомі гравці – Apple, Google та Microsoft, при цьому Google та Microsoft надають можливості для розширення своїх рушіїв за допомогою плагінів. Отже, раціональним виглядає не створення нових систем, а інтеграція з уже існуючими, оскільки вони надають достатньо можливостей для цього.

Отже, ми зробили припущення, що одним з варіантів вирішення проблеми із надто великою статичністю презентацій є створення анімацій, які будуть відображати ефект малювання або рисування зображення.

Мета роботи: створення плагіну для презентацій Google, що дозволяє перетворювати статичне зображення в анімацію, яка відображає процес малювання такого зображення.

Завдання:

1. Аналіз алгоритмів сегментації та заповнення зображення, що дозволяють імітувати процес малювання зображення.
2. Знаходження оптимального варіанту поєднання алгоритмів, що дозволить швидко та ефективно створювати таку анімацію.
3. Створення інтерфейсу користувача, який буде відповідати вимогам надійності та буде зручним для користувача (не буде перевантаженим великою кількістю зовнішніх параметрів).

4. Створення плагіну для презентацій Google, який буде надавати створений раніше інтерфейс для легкого використання на даній платформі.

Об’єкт дослідження: алгоритми сегментації зображень, платформа Google Docs Add-ons.

Предмет: плагін, що перетворює зображення в анімацію, яка відповідає ефекту малювання такого зображення.

Наукова новизна: дана робота є першою роботою, яка досліджує можливості створення зазначеної анімації не як окремого додатку, а як компоненту, що може бути інтегрованим в деякий рушій для презентацій.

Практичне значення роботи полягає у можливості застосування розробленого плагіна з метою створення більш динамічних презентацій.

Структура курсової роботи. Курсова робота складається з двох розділів та підрозділів, списку використаних джерел (9 найменувань), трьох додатків.

Аналіз алгоритмів сегментації та поступової появи зображення

Аналіз задачі імітації малювання об’єкта

Залежно від змісту зображення, його «природній» спосіб рисування відрізняється. Давайте проведемо класифікацію типів рисування за змістом зображення.

1. **Друкований текст.** Текст, що є друкованим, має з’являтися буква за буквою. У такому разі спочатку у зображенні виділяються певні кластери, які надалі і будуть відображатися один за одним.
2. **Рукописний текст.** Текст, що написаний від руки, має писатися слово за словом, зліва праворуч та зверху вниз. Це більш складна задача, ніж для друкованого тексту, адже відображення має бути плавним, передаючи природні рухи руки людини. Кожен сегмент не може бути розгалуженим та має заповнювати всю ширину штриха.
3. **Рисунок**. Відображення має відбуватися сегментами, схожими на штрихи, як і у випадку рукописного тексту. Відмінність полягає у порядку штрихів. У рисунку їх зображення є більш природнім за розміром (від більшого до меншого), а також за кутом.
4. **Малюнок**. Однозначного «природнього» алгоритму не існує. Може сприйматися як сукупність відтінків, які накладаються шар за шаром, як у випадку із малюванням аквареллю, або ж сукупність відтінків без накладання кольорів, як у випадку масляних фарб. Найбільш простим способом роботи є штрихування подібних кольорових областей. При цьому штрихи можуть бути довільно нахиленими і наноситися довільно, але при цьому вони мають бути значно ширшими, ніж у випадку рисування.
5. **Комбінації.** Малюнок містить одночасно і заповнені кольорові області, і штрихи; спочатку має застосовуватися рисування штрихів, а потім – заповнення форм. Відповідно, у такому разі має бути застосована комбінація із способів для малювання та рисування.

Для спрощення задачі, у даній роботі ми сфокусуємося на перших чотирьох способах. Кожен з них дає свою унікальну анімацію, при цьому ми відкинемо ускладнення, яким є комбінація декількох способів.

Аналіз задачі створення анімації малювання як комбінації алгоритмів з обробки зображення

Для способів рисування, що не включають в себе багатошарового накладання кольорів, загальний алгоритм виглядає наступним чином.

1. Розбити зображення на окремі ділянки, відповідно до їх змісту.
2. Використати алгоритм заповнення відповідно до типу зображення для кожної з таких ділянок.

Конфігурація ділянок є можливістю для визначення типу зображення. Наприклад, монотонні сегменти вказують на те, що перед нами рисунок, однакова середня висота ділянок – на те, що це текст, а відсутність подібних зв'язків вказує на те, що на обробку був надісланий малюнок. Загалом існує тенденція, що довгі та вузькі ділянки мають зарисовуватися так, як для рисунку. Що ж стосується ділянок між ними, їх можна спробувати приймати за фон зображення (у випадку схожості кольорів). У такому випадку власне фон має бути відображений до початку рисування, а фон під штрихами має бути апроксимованим від сусідніх ділянок. Якщо ж ділянки мають принципово різні кольори, вони мають бути зафарбовані за допомогою штрихів, що будуть залежати від специфіки роботи алгоритму, а не від конфігурації ділянки.

Також для спрощення ми надалі будемо розглядати лише монохромні зображення, щоб не ускладнювати алгоритм обробкою ситуацій із поєднанням кольорів.

Порівняння алгоритмів сегментації зображень

Спробуємо знайти найбільш зручні алгоритми сегментації зображення, що є водночас і ефективними за часом, і прийнятними за отриманим результатом.

Наведемо порівняння деяких з таких алгоритмів.

Назва	Принцип дії	Особливості	Складність
Region-based Segmentation	Вираховуємо середнє по трьом кольорам значення. Порівнюємо із середнім по всьому зображенню. Якщо менше – кластер 1, більше - кластер 2. Можна додати більше градацій для більшої кількості кластерів.	Тривіальна імплементація, неточні результати.	O(n)
Edge Detection Segmentation (Sobel operator)	Використовуються наперед визначені або ж обраховані вагові матриці. Накладаємо матрицю на частину зображення, множимо значення. Вираховуємо дискримінант отриманої матриці. Результат виражається коренем з суми квадратів дискрімінантів отриманих матриць. Далі можемо використати середнє значення для отримання границь.	Простий та досить швидкий. Не потребує попереднього з'ясування кількості кластерів. Дає не безперервні границі, тобто кластери неможливо чітко виділити. Досить низька точність.	O(n)
Canny algorithm	Використовується попередній алгоритм до кроку 5. Використовуються під-алгоритми non-maximum suppression та double threshold; Виокремлення границь за евристикою.	Дає достойний результат, зазвичай з неперервними границями. Залежить від вхідних параметрів, тому може дати забагато чи замало кластерів.	O(n), але набагато більша константа, ніж у попередніх алгоритмів.
Cycle algorithm	Випадковим чином обирається деяка точка.	Границі завжди закриті.	У загальному

	Рух завжди відбувається за контрасністю точок на границі. Рано чи пізно ми обходимо об'єкт по границі, отримуємо певну форму.	Від евристики визначення контрасності залежить те, скільки об'єктів ми знайдемо після множинного запуску.	випадку O(n), з набагато меншою константою.
Seeded region-growing	Виділяються точки (sees), навколо яких будуть формуватися регіони (випадковим чином). Використовується евристика для визначення приналежності до регіону сусідніх точок (зазвичай за інтенсивністю, вимірюється до однієї чи до багатьох точок). Приєднується до певного регіону	Важливо вгадати розташування seeds та їх кількість. Залежить від евристики та її точності.	O(n), з досить великою константою.
Unseeded region-growing	Вибирається певна точка. Якщо різниця значення евристичної функції більша за певне число T, вона відноситься до нового кластеру, якщо ні, то до старого.	Тривіальна реалізація. Не залежить від наперед обраних точок.	O(n)
K-Means clusteing	Обираємо декілька довільних точок. Створюємо кластери за відстанню до точки. Новими центрами стають точки з найменшими відстанями до кожної іншої точки. Перерозподілення точок	Потрібно наперед "вгадати" кількість кластерів. Потрібна попередня обробка зображення	O(S) – за сталої кількості ітерацій NP-повна задача (за обмеження схожості попереднього і поточного результатів).

Як можемо побачити, деякі алгоритми з наведених не є прийнятними у нашому випадку. Так, ті, які потребують вводу кількості кластерів або розташування ключових точок, у випадку плагіну для довільного користувача просто не можуть бути реалізовані. Тому ми маємо обрати ті алгоритми, які дають більш-менш рівномірний результат і потребують мінімальної кількості зовнішніх параметрів, такі як, наприклад, Canny algorithm або ж Unseeded region-grwoing.

Оскільки другим етапом у створенні нашої анімації є алгоритми заповнення визначених ділянок, тут одразу можна помітити, що нам добре підходять алгоритми із сімейства region growing. Вони дають покрокове нарощення ділянки, при цьому вже не є принциповим той факт, що алгоритм має наперед знати певні дані, такі як границі кластеру або ж точка росту, адже під час виконання роботи попереднього алгоритму ми можемо забезпечити алгоритму ці дані.

Підбір найкращої комбінації алгоритмів для роботи із різними типами зображень

Задачі, які постають перед нами під час розробки анімації, набагато складніші, ніж може скластися враження на перший погляд. Так, підібрані алгоритми мають вирішувати наступні проблеми:

1. Розбити зображення на сегменти, які мають представляти собою окремі об’єкти.
2. Визначити порядок замальовки таких об’єктів.
3. Визначити, що буде відображатися під цими об’єктами до замальовки.
4. Використовувати оптимізації, що допоможуть робити кроки, зазначені вище, за прийнятний час.

5. Оскільки алгоритм призначений для широкого використання, він має вимагати мінімального втручання зі сторони користувача.

Однак, дана задача пропонує і також дозволяє нам виконати деякі спрощення, що принципово не відобразяться на результаті роботи:

1. Сегментація важлива, але для нас не важливо, наскільки вона буде точною у масштабі окремих пікселів. Анімація має бути швидкою, отже користувач не помітить окремих дефектів зображення і вони ніяк не вплинуть на прикінцевий результат.
2. Порядок малювання також досить довільний. Для нас головне лише, щоб анімація виглядала «природньо», тобто для малювання під кутом, наприклад, для нас неважливо, чи будуть з'являтися деякі окремі кроки анімації, і їх можна пропускати.

Отже, для сегментації нам не потрібен абсолютно точний алгоритм, заснований, наприклад, на роботі нейромережі. Ми можемо використати простіші евристики.

Для сегментації ми використали алгоритм «водорозділу» ("watershed"). Він добре оптимізований і підходить у більшості простих випадків. Крім того, його реалізація не потребує наперед знати кількість об'єктів на зображенні, а цю інформацію ми не можемо надати. Коротко опишемо реалізацію цього алгоритму.

1. Зображення переводиться у чорно-білий формат, кольорові значення апроксимуються.
2. Беруться локальні мінімальні значення.
3. Від мінімальних значень починається розповсюдження областей до сусідніх пікселів.
4. Якщо під час розповсюдження на стику між двома областями досягається порогове значення, області розмежовуються.
5. Алгоритм зупиняється тоді, коли все зображення було розмежовано (не залишилося пікселів, на які можна розповсюдитись).

Зазвичай питання полягає у пошуку порогового значення, яке часто ставлять у залежність від інтенсивності зображення. Від цього значення залежить кількість областей, які будуть утворені.

На зображенням із середнім значенням 5-10 об’єктів, алгоритм дає нормальні результати, однак під час великого скупчення об’єктів різної середньої інтенсивності можуть виникати проблеми – потрібні значення просто не будуть знайдені.

Для того, щоб покращити роботу алгоритму, використовують попередню обробку, на кшталт розмивання Гауса.

Результат роботи алгоритму можна побачити на наведеному нижче прикладі.

Як ми можемо побачити, робота сегментації не ідеальна. Три з представлених камінців не були помічені алгоритмом і злилися з фоном. Однак, у випадку, коли об’єкти не настільки чітко відділені від фону (з точки зору людини), а становлять єдиний масив, алгоритм справляється краще, ніж інші. Оскільки алгоритм при його недоліках є досить швидким, ми можемо обрати його.

Наступне питання, яке ми маємо вирішити – налаштування фону зображення. Для його найпростішого вирішення можна апроксимувати значення пікселів навколо об’єкту, проте через це виникне пляма, якщо навколо були тіні чи інші об’єкти. Набагато кращим рішенням є апроксимація кольору фону (фон ми знаходимо на етапі сегментації). Також ми можемо надати колір фону на розсуд користувачеві (як опціональна можливість), адже вказати колір не є проблемою.

Для вирішення проблеми фону потрібен алгоритм заповнення під кутом. Для цього використаємо створений нами алгоритм:

1. Ітеруємо за кожним знайденим контуром.

2. Верхня ліва точка є кутом прямокутника контуру. Для кожної точки у межах прямокутника знаходимо її відстань до кута помножену на косинус кута, який утворюють відрізок між точкою та кутом із вертикальною прямою.
3. Розбиваємо відстань на порогові значення, створюємо анімацію, ітеруючи за пороговими значеннями та створюючи відповідні кадри.

Таким чином, для замальовки нам не потрібно кожного разу ітерувати за пікселями всього зображення, достатньо лише прямокутника, в якому знаходиться даний сегмент.

Така зв'язка алгоритмів буде добре працювати у випадку графічних замальованих об'єктів. У випадку креслень або ж літер нам не потрібен такий складний алгоритм сегментації – достатньо лише порогового значення із накладанням розмивання Гауса. Однак, у такому випадку нам вже не обійтися простим штрихуванням під кутом під час анімації.

Для рисування штрихів нам знадобиться власна імплементація алгоритму росту регіонів. Цей алгоритм схожий на описаний вище, однак тут ми будемо знати точки росту наперед – це зазвичай правий лівий куток.

1. Оберемо першу точку знайденого сегменту.
2. Знайдемо сусідні клітинки, за 8-зв'язністю.
3. Замалюємо сусідні клітинки сегменту.

Зазначимо, що наш алгоритм буде працювати «у ширину» (BFS), адже нам потрібно заповнювати штрихи більш-менш послідовно.

Таким чином, ми вирішили ключові проблеми нашої системи.

Розробка плагіну для створення анімації

Вибір концепції роботи плагіну

Під час підготовки до роботи ми планували створити анімацію для Google презентацій або ж Microsoft PowerPoint, яка буде спиратися на стандартну

схему анімацій. Однак, стало зрозуміло, що це не спрацює, оскільки обидва ці рушії не надають інструментів для створення нативної анімації.

Найпопулярнішим форматом для збереження анімації є GIF. І хоча зазвичай ми звикли до його зацикленої форми, яка не підходить у нашому випадку, у ньому можна створити і одноразову анімацію. Завдяки своїй розповсюдженості, операції із об'єктами цього формату підтримує велика кількість бібліотек.

Отже, ми плануємо створити плагін, що буде генерувати незациклену GIF-анімацію, яка надалі може бути вставлена на слайд. Для поступової появи елементів слайду можна використовувати стандартні анімації рушія, тому що перше виконання GIF буде робити після своєї появи на екрані.

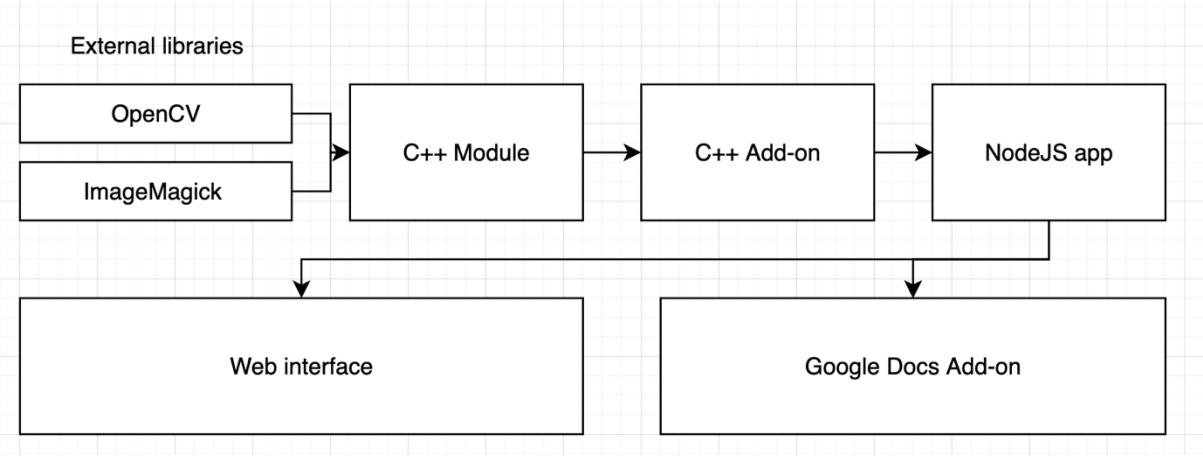
Вибір технологій для створення анімації

Оскільки операції над зображеннями не є тривіальними та мають задіювати декілька складних бібліотек, правильно буде делегувати їх серверній частині додатку. На сервері ми можемо легко розгорнути найбільш потрібні бібліотеки для обробки зображень, такі як OpenCV для створення окремих шарів та ImageMagick для створення анімації. Ці бібліотеки мають API для різних мов програмування, проте для швидкості роботи, яка є принциповою для нашого проєкту, ми оберемо C++. Ця мова програмування надає зручні можливості попіксельної обробки зображень, що є надто повільною в таких мовах програмування, як Python.

Для створення зручного в управлінні серверу ми оберемо Node.JS. Цей фреймворк є новим та досить популярним сьогодні, але найбільше нас зацікавила його функція C++ add-ons. Вона дозволяє нативно інтегрувати в цей рушій програми на C++, що знадобиться в нашому проекті.

Для розміщення серверу з такою непростою структурою залежностей було використано VPS. Що ж стосується поєднання серверу із презентаціями

Google, існує єдина можливість для цього – стандартний набір інструментів Google Add-ons. Отже, наведемо схему, яку ми будемо використовувати для розгортання нашого продукту.



Створення абстрактного інтерфейсу користувача

Під час створення інтерфейсу користувача ми маємо ставити перед собою наступні цілі:

1. Зробити інтерфейс зрозумілим та змістовним для користувача.
2. Дати можливість користувачеві змінювати поведінку застосунку у межах наданих алгоритмів.

Під час розробки алгоритму ми напрацювали два режими його роботи – для рисунків та малюнків, які принципово різняться за принципом роботи. Також для того, щоб запобігти невдалій апроксимації, ми можемо дозволити користувачу задавати колір фону. Для наочності ми також можемо створити дві версії анімації. Перша буде програватися в нескінченному циклі для демонстрації роботи, а друга буде зупинятися після першої ітерації, для вставки у презентацію.

Технології Google Docs Add-ons для створення плагінів Google Docs

При створенні плагінів для Google Docs використовуються HTML, CSS та Google Script для «бекенду» плагіну (видозмінена версія JavaScript). На

фронтенді ж застосовується звичайний JavaScript. Оскільки наша задача не потребує використання GoogleScript, зосередимо увагу на фронтенді. Загалом програмування плагіну – це програмування, схоже на будь-яку іншу веб-верстку. Головна особливість – це наявність спеціальної бібліотеки, яка містить стандартні для екосистеми Google елементи управління. Для проходження верифікації та потрапляння у магазин доповнень також слід дотримуватися стандартів Google з обробки персональних даних та з дизайну. В нашому випадку ми не використовуємо збір даних крім як за згодою користувача (завантаження зображення), тому для нас це не стає проблемою.

Також Google надає інструментарій для роботи з плагінами. Під час розробки ми можемо використовувати власний документ для тестування.

Висновки

У рамках даної роботи було виконано декілька важливих завдань. Ми проаналізували поставлену задачу та розділили її на складові частини. Ми побачили складність проблем, які стояли на шляху анімації рисування та спробували знайти оптимальне рішення для кожної з них. Були проаналізовані деякі алгоритми сегментації зображень та обраний найбільш вдалий з них для кожного з режимів роботи.

У другому розділі ми спроектували архітектуру плагіну для Google Docs, який задовольняє потреби в мінімалізмі та ефективній передачі даних на сервер, а також окреслили загальну структуру взаємодії сервера та клієнта.

Основною частиною роботи стала імплементація алгоритма та налагодження передачі даних між всіма частинами системи – C++ модулем та його бібліотеками, NodeJS, VPS та Google Add-ons.

У рамках виконання роботи виникли деякі труднощі. Недостатня реалістичність анімації, недостатнє розпізнавання деякої частини об’єктів та проблеми підбору оптимальних порогових значень становлять значний виклик у роботі із растровими зображеннями. Для того, щоб просунутися у цьому напрямку, слід спробувати створити більш узагальнений алгоритм, що буде враховувати як наявність штрихів, так і існування замальованих областей та опрацьовувати обидва випадки належним чином.

Таким чином, ми виконали основні завдання, поставлені на початку роботи, а також довели твердження про можливість створення генератора анімацій малювання зображення.

Список джерел

1. https://www.labnol.org/internet/write-google-docs-addon/28446/ – How to write Add-on for Google Docs.
2. https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html – Image Segmentation with Watershed Algorithm
3. https://nodejs.org/api/addons.html – NodeJS C++ add-ons.
4. https://computer-vision-talks.com/post/2013-08-19-cloud-image-processing-using-opencv-and-node-js/ – Cloud image processing using OpenCV and Node.js
5. https://d1wqtxts1xzle7.cloudfront.net – High Level Computer Vision using OpenCV