# Описание на проблема/задачата

Задача е да се направи реализация на CNC Writing machine, която ще наричаме плотер. Плотерът е изходно периферно устройство, което дава възможност за извеждане на графична информация върху хартия.[1]

Като вход програмата трябва да получава изображение. Това изображение след това трябва да бъде обработено до формат, в който информацията ще може да се изчертае от плотера.

Плотера ще може да черта само в един цвят.

# Решение и теоретична обосновка

Основно задачите на плотера могат да се разделят да следните основи задачи:

* Проектиране и физическа реализация
* Прочитане и обработка на изображението
* Изчертаване на изображението

### Физическа реализация

Проектът се състой от две основни части – Обработка на изображението и изчертаване на изображението.

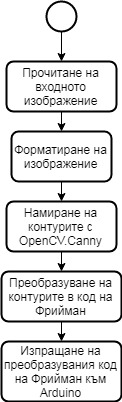
Обработката на изображението се осъществява с помощта на компютър с операционна система. Като резултатното изображение се предава по серийна комуникация към Arduino през USB.

Arduino чрез серийна комуникация получава изображението на база, на което подава команди към двата стъпкови мотора(X и Y) и серво мотора(химикалката – Z).

Реализацията на местенето по осите е само чрез местене на пишещата глава по трите направление чрез винтово задвижване.

### Прочитане и обработка на изображението

За прочитане и обработката на изображението ще се използва езика Processing[2]. Този език е избран, защото има изградени много функционалности свързани с комуникацията между Processing и Arduino. Допълнително ще използва библиотеката OpenCV for Processing[3], в която са имплементирани основните алгоритми за обработката на изображението, които ще бъдат използвани. Биоблиотеката не е развита напълно и възможностите на OpenCV липсват или са ограничени.



фигура Описание на стъпките

Форматирането на изображението включва промяна на размерите и ориентацията на изображението, ако размерите са по-големи от размерите на плотера.

За намиране на контурите се използва алгоритъма Canny edge detector, защото е широко приложим за различни изображения. Това е многостъпков алгоритъм, който удовлетворява три основни критерии[4]:

* Ниско ниво на грешка – добро намиране само на съществуващите контури
* Добро локализиране – разстоянието от истинския контур до намерения да бъде минимално
* Minimal response: Only one detector response per edge.

Стъпки на Canny edge detector[4][5]:

* Филтриране на шума – За тази цел се използва Гаусов филтър.
* Намиране на интензитния градиент на изображението – За тази цел се използва алгоритъма Sobel, който определя контурите по ширината и дължината.
* Потискане на не-максимумите – По този начин се премахват пиксели, които не трябва да бъдат част от контура.
* Определяне на финалните контурите – Използват се два параметъра upper и lower threshold.

Canny edge detector не е прост оператор, което го прави по-тежък за изпълнение в сравнение със Sobel, който е част от самия алгоритъм и други алгоритми за намиране на контури. Предимството е,че премахването на шумовете, многостъпковото определяне на контурите и премахването на излишните такива.

В библиотеката OpenCV for Processing има възможност единствено за определянето на параметрите upper и lower threshold. Според препоръките на Canny тяхното отношение трябва да бъде около 2:1 и 3:1[5].

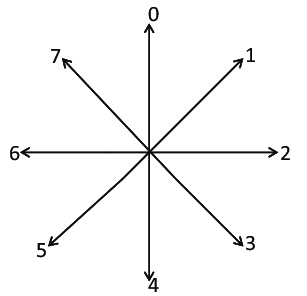
В имплементацията се използва статистически подход на база стойностите на пикселите в изображението[6]. Направено е сравнение на определяне по средна стойност и медиана. Също се използва подход с предварително изравняване на хистограмата. Определената стойност от медиана се взимат съответно с коефициенти 0.66 за долната граница и 1.33 или 1.98 (според препоръките за отношение 1:2 и 1:3) за горна граница.

След опити с много различни изображения се установи, че използваната имплементация на Canny не се справя добре с области, в които има дребни и рязко сменящи си посоката граници(например ситно назъбени стръкове трева). Също примерите изложени в [6] не постигат същата добър резултат. Това най-вероятно се дължи на различните имплементации на Canny и подавани параметри по подразбиране на отделните стъпки.

//TODO: Да се сложат примери за лошо получени крайни резултати

//TODO: Добавяне на анализ на другите изображения

Следващата стъпка е контурите на изображението да се изпратят на Arduino за изчертаване. Използва се Chain Freeman Code с 8 посоки. Този алгоритъм връща като резултат последователност от стойности, който оказват къде се намира следващата точка от контура спрямо текущата. Стойностите са от 1 до 7 съответно:



фигура Координати на Freeman code

Алгоритмът за обработка и предаване на комуникацията от Processing на Ardiuno:

1. Последователно се итерира до пикселите на изображението
2. Ако се стигне до черен пиксел(т.е. пиксел, който има граница)
   1. Подават се последователно координатите на този пиксел на Arduino
   2. Подава се команда за сваляне на писеца
   3. Пиксела се маркира като бял, за да не бъде повече обработван
   4. Ако има намерен съседен пиксел, който е част от контура:
      1. Намиране на код на Фрийман за следващия пиксел от контура
      2. Изпращане код на Фрийман за контура
      3. Връщане на стъпка 2.c.i с текущ пиксел намерената следваща точка от контура.
   5. Иначе
      1. Подава се команда за вдигане на химикала
   6. Връщане към стъпка 2

В Arduino, при прочитане на стойност, ако химикалката е вдигната, то се получават координати за началото на следващия намерен контур.

След прочитането на две последователни стойности, трябва да бъде получена команда за пускане на химикалката.

Следващите получени стойности са код на Фрийман за описване на контура. Изчисляват се новите получени координати на химикалката и след това се подава за преместване към стъпковите мотори.

Това се прави докато не се получи отново команда вдигане на химикалката.

# Програмна имплементация на решението

Описанието на проблема следва последователността и логиката на имплементацията на програмата. Пълният код на програмата може да бъде намерен в [GitHub](https://github.com/ElitsaVenchova/Plotter).

# Референции

[1] [Wikipedia/Плотер](https://bg.wikipedia.org/wiki/%D0%9F%D0%BB%D0%BE%D1%82%D0%B5%D1%80)

[2][Processing Official Site](https://processing.org/)

[3][GitHub/OpenCV for Processing](https://github.com/atduskgreg/opencv-processing)

[4][docs.opencv.org/canny\_detector](https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html)

[5] <https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html>

[6] <http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/>