**Logo-TU-black-3DТехнически университет – София, Филиал Пловдив**

**Факултет по Електроника и Атоматика**

**Катедра Системи за управление**

**Специалност: АИУТ**

Курсов проект

**Тема: Управление на мобилен робот с визуална обратна връзка**

Изготвил: Проверил:

Симеон Марлоков /доц. Н. Шакев/

АИУТ IV курс

Фак. №366750

СЪДЪРЖАНИЕ:

I. Въведение 4

II. Python и мястото му в новата революция 7

III. Мобилни роботи 8

IV. Хардуер на проекта 10

V. Реализация 11

VI. Бъдещо развитие 12

a. Приложение 13

 Библиография: 13

 Хардуерна спецификация: 14

 Програмен код – Визуален модул: 17

1. Въведение

За начало на Вселената се счита Големият взрив[1], който е възникнал преди 14 милиарда години. Благодарение на него започват да се образуват всички космически тела, включително и нашата галактика – Млечния път. Времето тече, а с него идва и момента на появата на Слънчевата система и преди 4 милиарда и половина години се образува планетата Земя, на която живеем ние, човешката раса, появили се преди около 200 хиляди години. Нашето същестуване е като капка в океана на времето, но в кратката за Вселената и нашата Земя история ние, човечеството, правим повратният си момент в избора на интелекета. Това е път, по който веднъж тръгнали след последната Ледникова епоха никога няма да можем да спрем да следваме, ако искаме да се развиваме като раса. С нищо по-добри от другите живи организми на планетата, живеещи в променящата се флора и фауна, ние приспособяваме средата за нас, отколкото себе си за нея, с единствените оръдия, които имаме над другите организми – интелект и силно развиваща се и организираща се общност. Но и за това е трябвало време – цели 190 хиляди години. Тогава откриваме земеделието и с това слагаме край на инфантилния си период. Хората за първи път имаме начин да отглеждаме култури, а не да ловуваме и берем. Това само по себе си налага първите трайни заселвания и създаване на големи общности – първите градове.

От този момент човечеството минава през много епохи в кратката си съзната история, някои ясно дефинирани и други успорвани до ден днешен, но с една тенденция – ускоряване. Всяка епоха и етап в развитието на човека носи скокообразно развитие в нивото на обществото и технологията – цивилизацията като цяло. Така достигаме до 19 век от новото време, когато човечеството навлиза в Първата индустриална революция. За първи път във вече 199 хиляди годишната ни история предстои да направим поврат в целия си начин на живот и на планетата. Чертите на революцията са навързаността между отделните технологии, които се появяват в периода и капацитета им да променят не само икономиката, но и неизбежно обществото. Всичко се случва с рязко създаване, апликация и дифузия. Ефективност и продуктивност са първите ключови думи за икономиката, а за човека в общността са удобство и възможности. Появяват се първите фабрики. Масовост. Производство. Образование – за първи път човешката раса като едно цяло започва да се образова, а не само избрани слоеве на обществото. Замърсяване – първите проблеми в световен мащаб около *масовата* вече употреба на въглища - не винаги една революция носи положителни белези със себе си.

Погледнато обзорно, най-важните революции в последните столетия са:

* Първа индустриална революция – 1780г. – 1840г.
* Втора индустриална революция, Техническата революция – 1870г. – 1920г.
* Трета индустриална революция, Дигиталната революция – 1975г. – днес

Обаче не е време да гледаме назад, тъй като отново сме в навечерието на следващата революция – четвъртата индустриална революция или още Индустрия 4.0 и Общество 5.0. Това е първият случай на революция директно продължаваща в стъпките на предходната.

През 2015г[2]. Клаус Шваб[3], немски инженер и икономист роден през 1938г., за първи път въвежда този термин, който по-късно ще се превърне в „Покоравяне на Четвъртата индустриална революция“, темата на Световния симпозиум по икономика през 2016.

*„Размерът, обхватът и сложността на тази трансофрмация ще бъде такъв, какъвто човечеството никога не е виждало до сега. Революцията ще промени фундаментално принципите по които живеем, работим и комуникираме един с друг. Едно е ясно – ответният ни отговор трябва да е интегриран и всеобхватен, обвързващ всички – от публичния и частния сектор, от академичния до цивилния.“ Klaus Schwab, The Fourth Industrial Revolution, Foreign Affairs, December 2015*

Ключовите думи на новата революция са различни спрямо сферата в която тя се разглежда. За обществото това е възможността хората да комуникират един с друг без границите на времето и географията, да имат достъп до цялото човешко познание. Зa индустрията това е възникването на нови технологии и пълна автоматизация, изразяваща се в роботизираното производство куплирано с интернет на нещата, умни системи за контрол, наблюдение и поддръжка прерастващи в умни асистенти и изкуствен интелект, квантово изчисляване, триизмерно принтиране, нанотехнологии и други. Всичко това дава възможност на човечеството да повиши своя доход и подобри начина си на живот в глобален аспект. В техническата индустрия това е по-добра логистика, по-добро и бързо производство, преустановяване на човешкия фактор (замяната на ниско квалифицирания труд с роботизиран такъв), пад на цените и появата на нови пазари и общодостъпни възможности.

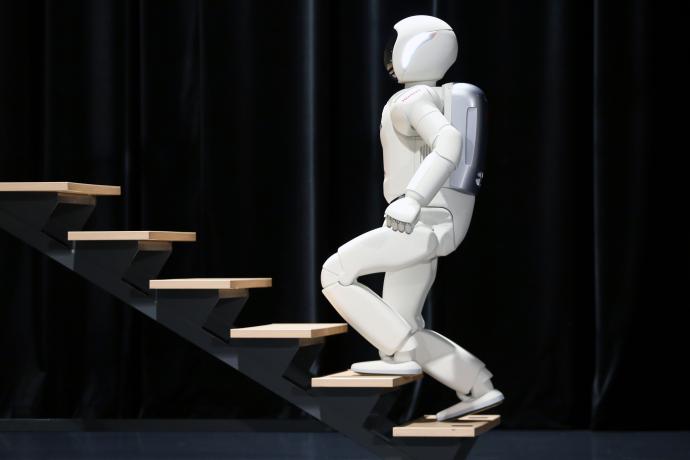


Figure 1: "Следващата стъпка напред" - хуманоиден робот Азимо

За целите на този курсова работа ще се спрем на роботиката, компютърното зрение и умните системи и тяхното приложение в новата индустрия.

1. Python и мястото му в новата революция

Създаден през 1990г. от Гуидо ван Росум, датски програмист, Python е интерпретиран програмен език от високо ниво, който днес се използва масово в академичните и индустриални сфери. Основната характеристика зад дизайна на езика са неговата структурираност и четливост и възможността за приложение в малки и големи мащаби – лесно може реализиран чрез Python проект да бъде мащабиран до индустриални размери. Освен това скоростта на изпълнение на командите е добре оптимизирана за повечето компютърни платформи, на първо място от които са дистрибуциите на Linux. Python може да се изпълнява и от ARM базирани микрокомпютри, още наречени системи на чип SoC.

Ключовият диверсифициращ фактор за езика е неговият отворен характер. Тъй като е създаден с идеята да е open-source (обществено достъпен) няма организация определяща строго достъпа и възможностите на езика. Всеки може да напише библиотека за него, а добавяйки лесното програмиране на езика и вече съществуващите базови библиотеки при компилация, не дълго след появата си многобройни програмисти и учени прехвърлят своите трудове на Python. Това спомага за създаването на многобройни и многофункционални библиотеки. Точно този инструментариум определя приложението на Python в новата индустриална революция като средство за изграждане на системи за машинно обучение, дълбоко обучение, компютърно зрение

1. Мобилни роботи

Двуколесните електромобили са клас моторни средства, които в себе си включват голям набор от членове – скутери, електрически колелета и много други техни разновидности като hoverboard и електромобили. Тяхната основна характеристика е наличието на едно или две електрозадвижвания и направляващо колело. Всички споменати до тук представители няма да се разглеждат в този проект, а под двуколесни електромобили ще разбираме т.н мобилни роботски платформи или на кратко роботи.

Роботите представляват вид електромобил, намиращ приложение в училищните и университетските среди като средство за обучение на ученици и студенти върху основните принципи на инженерно проектиране, програмиране и управление. Използват се масово в глобален аспект за състезания по роботика в различни категории и предизвикателства. Ключовият момент при създаване на робот стои в избор на неговото предназначение – за коя от категориите състезания ще се проектира. В зависимост това се подбират необходимите сензори и мотори за създаване на електрозадвижването на робота. Основните категории и характеристиките им са:

Следене на линия – роботите тук са леки и се състезават за постигане на рекордно време за преминаването на определено трасе. Моторите използвани за създаване на електрозадвижването им се характеризират с по-малко отношение на редукция, тъй като за тези роботи скоростта, съответно оборотите, са по-важни, от колкото въртящия момент. Разновидности на тази категория, запазващи конструкционните особености са суперлиния, лабиринт по линия и 3D лабиринт.

Мини симо – два робота се изправят един срещу друг на арена, тепих, като целта е единият да бъде избутан. За разлика от първата категория тук се използват различни мотори в зависимост стратекията на инженера. Използването на мотори с висок коефициент на предаване означава голям въртящ момент, което ще позволи на робота да избута противника, но пък ще намали скоростта му за реакция. Интересни комбинации тук включват регулируеми мотори както и мотори управлявани с по-високо от номиналното напрежение (т.н овърклок на мотора). Това дава възможност за изключителна скорост и при добра конструкция висок шанс за победа, но носи нестабилност на системата и при продължителна битка между роботите ще доведе до изгаряне на моторите.

Както бе споменато по-горе, от гледна точка на задвижване, по-голямата част от роботите се базират на триколесна система. Стандартно имат два мотора, ляв и десен, и направляващо колело, още наречено трета упора. Направляващото колело има функция на допълнителен баланс и упора, като премества центъра на тежестта на ново място. Това е изключително важно за правилното управление на робота, тъй като висок център на тежестта води до появяване на смущения по време на работа под формата на разбалансиране/клатене на робота. В зависимост от конструкцията на роботската платформа и нейната тежест се подбира съответстваща по размер и маса трета упора.

В зависимост категорията на робота се изпозлват различни сензори за осъществяване на обратната връзка към моторите. Основните типове сензори са за разстояние, най-често срещани под формата на оптронна двойна(инфрачервени излъчвател и приемник, поставени в корпус и механически разделени), ултразвуков и лазерен сензор. Заедно с тях се използват различни закони за регулиране – ПИ и ПИД закони.

За целите на този проект няма да използваме този набор от сензори, а ще базираме управлението на визуална обратна връзка от камера модул, намиращ се на корпуса на робота.

1. Хардуер на проекта

За платформа на проекта от хардуерната страна се използва AlphaBot[4] Seeed мобилна роботска платформа. Тя представлява лесно модулно решение, което бързо да може да бъде изградено и използвано за научни цели. Ключовите елементи са наборът от различни видове сензори за автономно движение, както и вградена съвместимост с Raspberry Pi 3 и камера за компютърно зрение.



Figure 2: Роботска платформа AlphaBot Seeed

За контролер на робота се използва Raspberry Pi 3 Model B+[5], ARM базиран микрокомпютър.

1. Реализация

Централно за системата е управлението чрез визуалната обратна връзка. За да може тя да бъде реализирана трябва по някакъв начин да бъде интерпретиран видео каналът и преобразуван в действителен управляващ сигнал към моторите.

Първо ще започнем с взимане на видео канал от камерата. Инициализираме камерата и гледаме кадър по кадър видео канала. Това се реализира по следния начин:

# Capture from camera at location 0

cap = cv2.VideoCapture(0)

# Capture frame-by-frame

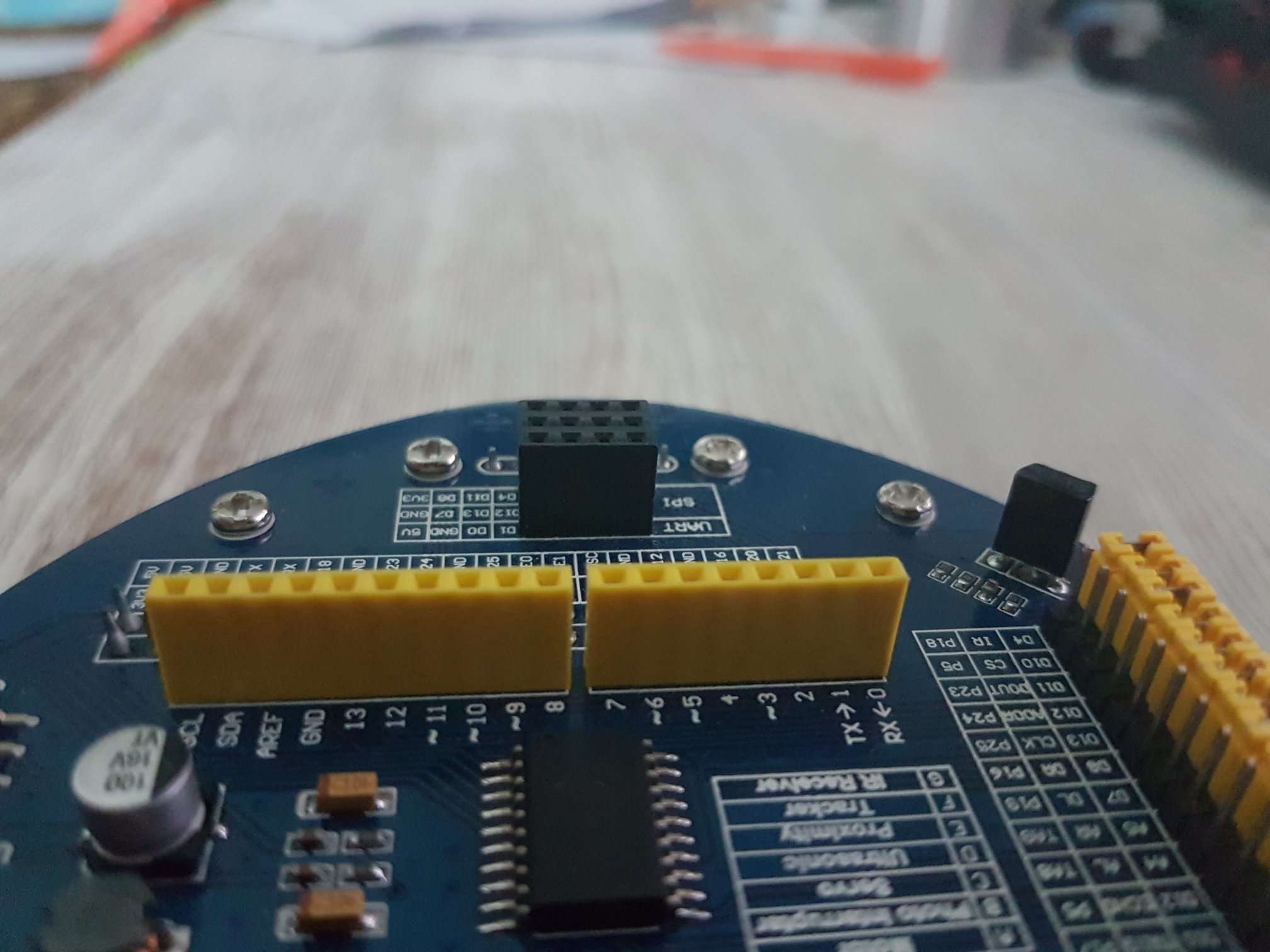
ret, frame = cap.read()

Figure 3: Поглед от камерата

Тъй като цветовете в стандартната RGB цветова система създават проблем при обработката в компютърното зрение трябва да преобразуваме изображението или в монохроматично, или в HSV цветова система. За целите на проекта ще използваме монохроматично изображение, тъй като в последващите обработки лесно може да се сведе до бинарно.

Превръщането в монохроматично изображение става чрез следната операция:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

За да може да се реализира в реално време алгоритъма при груб терен ще заличим част от изображението, за да намалим нужните изчисления по-нататък. Нуждата от това може да се елиминира ако се използва някакъв вид водач/маркер, като черна лента за отбелязване на пътя на робота.

Figure 4: Сиво/монохроматично изображение

Тъй като камерата ни носи много операция за белезите на пътя ще трябва да ги филтрираме с цел изглаждането им. Трябват ни само най-ярките белези – ръбовете. Има множество филтри, но за целите на този проект ще изберем билатерален филтър:

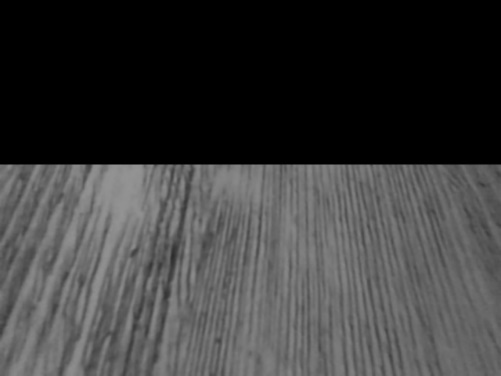
gray\_bilateral = cv2.bilateralFilter(gray, 5, 50, 50)

Figure 5: Филтрирани черти

След като част от белезите са залечени е време да пристъпим към разпознаване на ръбовете в картината. Има различни методи за тяхното разпознаване като Лаплас, Собол и изполвания тук метод на Канни. Всички се базират на свой математически принцип, като за Канни това е градиента – дали стойността на дадения пиксел се разполага в рамките на минимума и максимума.

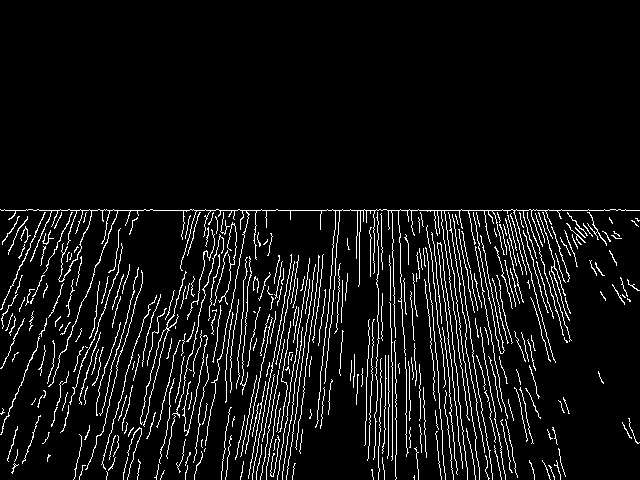
edges\_bilateral = cv2.Canny(gray\_bilateral, 40, 70)

Figure 6: Разпознати ръбове

Минусът на метода до тук е нуждата от добри нива на осветеност за правилна реализация на алгоритъма.

За следващата стъпка ще трябва да изчертаем фигура, представляваща „трасето“ по което се движи робота. Това става с т.н Convex Hull – когато имаме даден брой точки това е най-малката фигура, която обгръща всички точки. Ако трябва да разгледаме принципа с аналогия – ако имаме дъска с пирони и поставим ластик около всички тях, който да ги обгърне, то това е най-малката фигура в която те могат да се поберат.

#Convex Hull

CHull = edges\_bilateral #Giving CHull the image to compute

im2, contours, hierarchy = cv2.findContours(CHull, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# create hull array for convex hull points

hull = []

# create an empty black image

drawing = np.zeros((CHull.shape[0], CHull.shape[1], 3), np.uint8)

centers = np.zeros((CHull.shape[0], CHull.shape[1], 3), np.uint8)

# calculate points for each contour

for i in range(len(contours)):

# creating convex hull object for each contour

hull.append(cv2.convexHull(contours[i], False))

# draw object centers

cnt = hull[i]

M = cv2.moments(cnt) #Moment of the figure

if M["m00"] != 0:

Cx = int(M["m10"]/M["m00"])

Cy = int(M["m01"]/M["m00"])

cv2.line(centers,(Cx-3,Cy),(Cx+3,Cy),(0,255,0),3)#horizontal axis

cv2.line(centers,(Cx,Cy-3),(Cx,Cy+3),(0,255,0),3)#vertical axis

else:

# Unidentified exception

cX, cY = 0, 0

#cv2.imshow("Centers",centers)

# draw contours and hull points

for i in range(len(contours)):

color\_contours = (0, 0, 255) # red - color for contours

color = (255, 0, 0) # blue - color for convex hull

# draw ith contour

cv2.drawContours(drawing, contours, i, color\_contours, 1, 8, hierarchy)

# draw ith convex hull object

cv2.drawContours(drawing, hull, i, color, 1, 8)

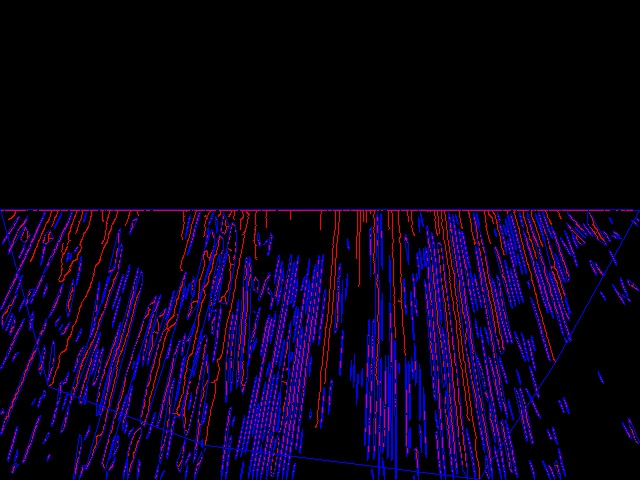
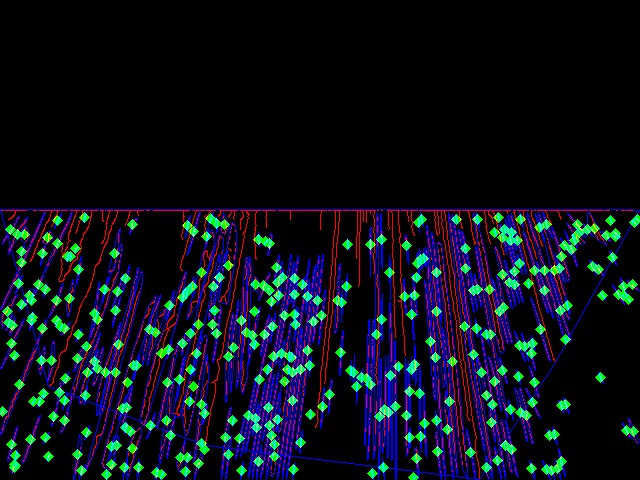
Както може да се види, процесорът е изчислил контурите и обгръщащата линия. Това вече дава предпоставка за изчисляване на центъра на обекта. Операцията за това е внедрена в кода горе.

Figure 7: Convex Hull на изображението

Виждаме всеки център в изображението като зелен маркер. В случая, тъй като не е подбран достатъчно силен филтър за грапава дървена повърхност виждаме центровете на всички разпознати фибри. Ако това се приложи в производствено помещение няма да имаме този проблем.

Figure 8: Центрове на ръбовете

За да можем да осъществим управлението, ще приемем, че механично роботът е в равновесие и ако прокраме мисловна линия през центъра на виждането от камерата изображение, то тя ще е перпендикулярна на пътя. А ако не е – означава, че имаме отклонение и трябва в зависимост на центъра на изчертаната фигура от пътя да коригираме подадените към моторите стойности.

1. Бъдещо развитие

С разрастването на индустрията и нарастването на ROS базираните системи с цел по-нататъчно развитие на проекта роботът може да се мигрира към ROS, като това ще позволи използването на по-интелигентна система за визуално управление – Visual Servoing Platform, ViSP. Това са библиотеки, създадени специфично за визуално управление на роботи. Обедидени заедно с AprilTag маркери, това ще донесе по-добра точност и по-висока скорост в реално време.

За по-добра сигурност на системата може да се добави модул, който да избяхва колизии. Това ще стане на същия принцип, като се използва за репер синия цвят на платката за изчертаване на контур.

1. Приложение

* Библиография:

[1] Теория за Големия Взрив– <https://en.wikipedia.org/wiki/Big_Bang>

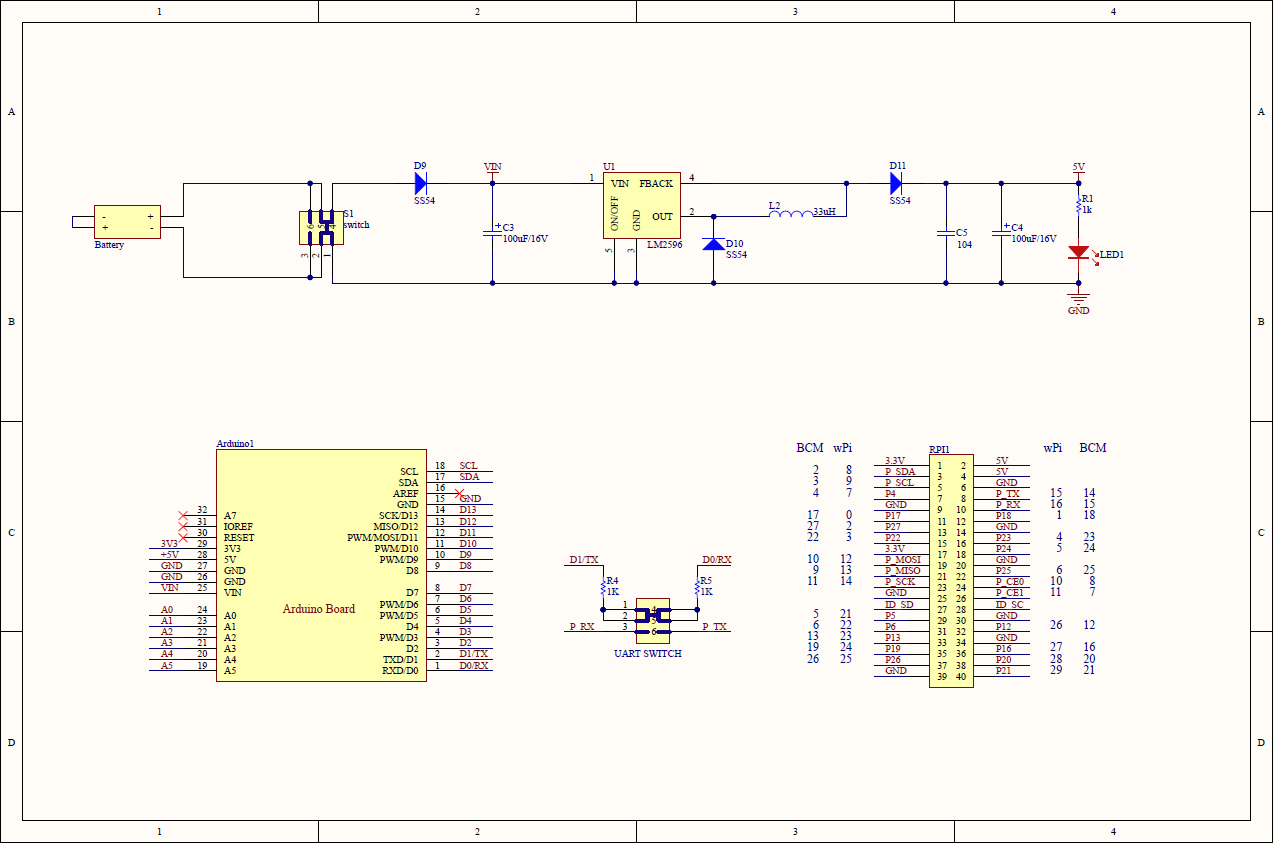
[2] Клаус Шваб– <https://en.m.wikipedia.org/wiki/Klaus_Schwab>

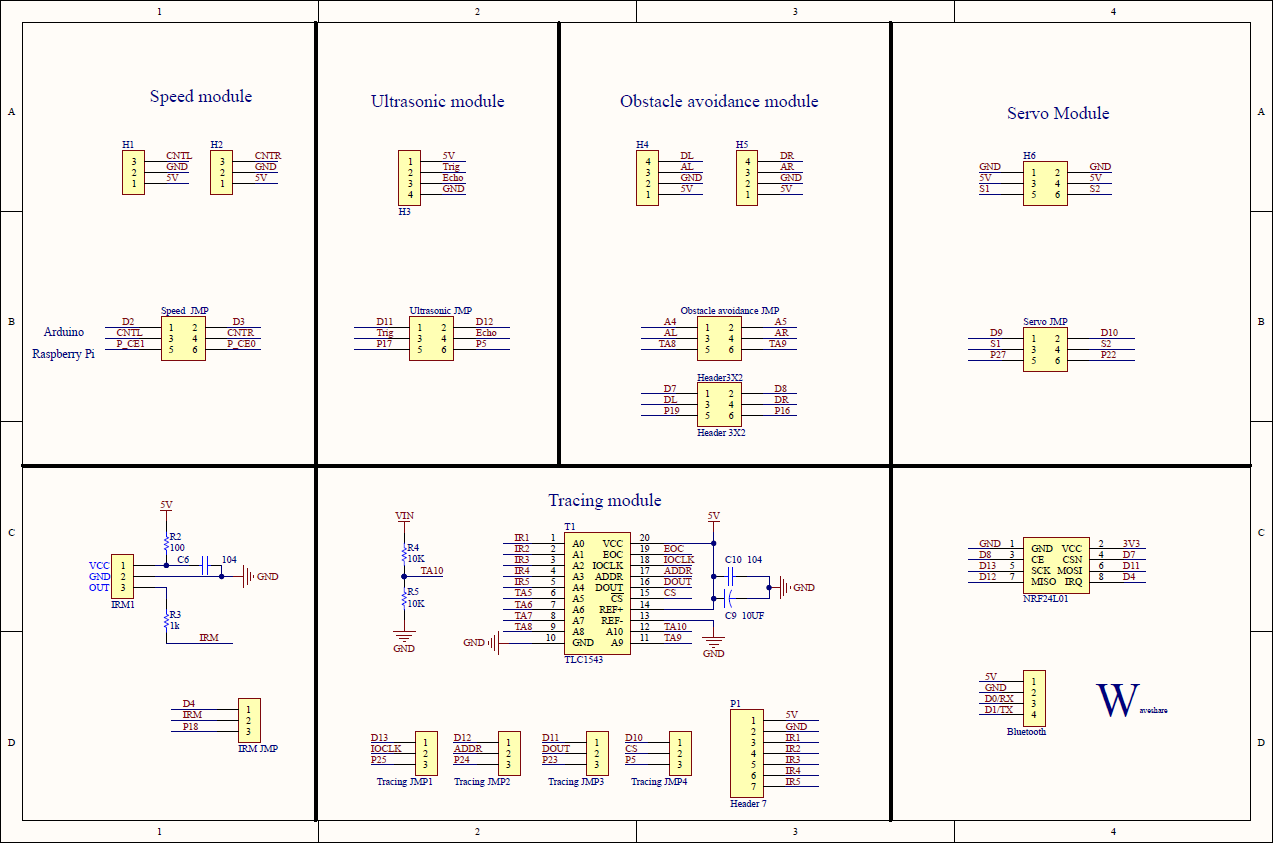
[3] Mastering the Fourth Industrial Revolution– <https://www.foreignaffairs.com/articles/2015-12-12/fourth-industrial-revolution>

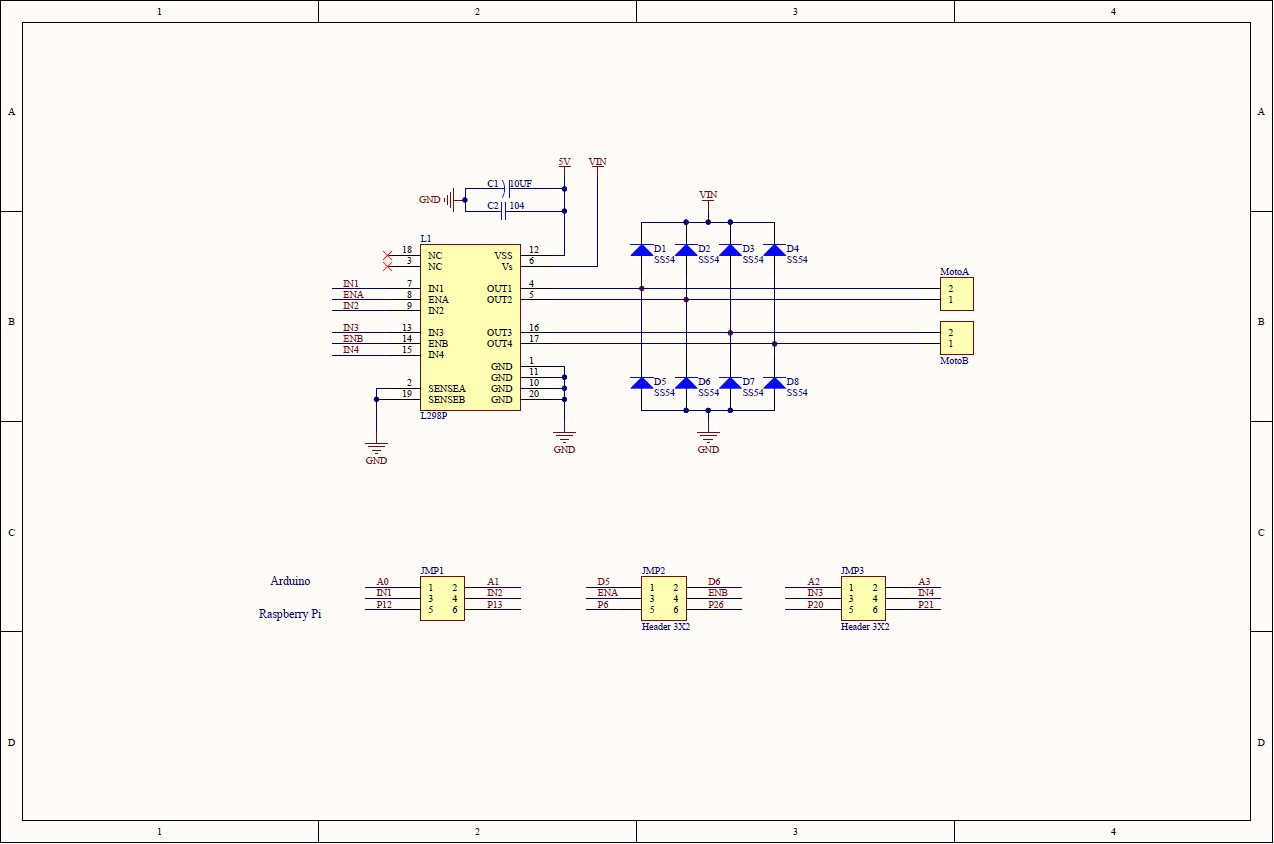
[4] AlphaBot Seeed - <https://www.waveshare.com/wiki/AlphaBot#Software>

[5] Raspberry Pi 3 Model B+ - <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>

* Хардуерна спецификация:







* Програмен код – Визуален модул:

# Vision Model

# @brief Init the video stream

# @param NA

# @return NA

import numpy as np

import cv2

import sys

def ImageZero(img):

for i in range(img.shape[0]-270): #Image width

for y in range(img.shape[1]): #Image hight

img[i,y] = [0,0,0]

def BlackExtractor(img):

for i in range(img.shape[0]):

for i in range(img.shape[1]):

if img[i,y] == [0,0,0]:

img[i,y] = [255,255,255]

else:

img[i,y] = [0,0,0]

# Capture from camera at location 0

cap = cv2.VideoCapture(0)

# Check hardware settings

#mycheck=cap.get(3) # - checks width

#print (mycheck)

#mycheck=cap.get(4) # - checks height

#print (mycheck)

#mycheck=cap.get(5) # - checks fps

#print (mycheck)

# 1 - Personal Camera - Logitech C220

Cam\_Width = 640.0

Cam\_Height = 480.0

Cam\_FPS = 30.0

# Set the width and height

cap.set(3,Cam\_Width) #width

cap.set(4,Cam\_Height) #height

cap.set(5,Cam\_FPS) #fps

# Check Stream

if not cap.isOpened():

raise Exception("Could not open video device")

sys.exit()

#BG Subtractor

fgbg = cv2.createBackgroundSubtractorMOG2(

history=10,

varThreshold=2,

detectShadows=False)

#Single Image Tester

#SingleImg = cv2.imread('Test.jpg',0)

while(True):

# Capture frame-by-frame

ret, frame = cap.read()

# Saving image from cam

#cv2.imwrite('Test.jpg',frame)

# Removing part of the image

ImageZero(frame)

#cv2.imshow("Tester",frame)

# Our operations on the frame come here

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Smoothing without removing edges.

# <frame, D(neighboring pixel count), treshhold> - a D>5 will not be practical for real

# time application on current RasPi.

gray\_bilateral = cv2.bilateralFilter(gray, 5, 50, 50)

# Testing with Gaussian

# Gaussian is too strong and features are erased. Must be in a well lit

# calibrated environment if used.

# gray\_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)

# Using the Canny filter to get contours

edges = cv2.Canny(gray, 20, 30)

edges\_bilateral = cv2.Canny(gray\_bilateral, 40, 70)

# Testing the Canny filter with different parameters

edges\_high\_thresh = cv2.Canny(gray, 50, 90)

def Extractor():

#@brief Extracts the background from the moving image parts. Needs the camera

#stream as input and returns the cropped stream

#@param capstream

#@return cropped

# Extract the foreground

edges\_foreground = cv2.bilateralFilter(gray, 9, 55, 55)

foreground = fgbg.apply(edges\_foreground)

# Smooth out to get the moving area

kernel = np.ones((50,50),np.uint8)

foreground = cv2.morphologyEx(foreground, cv2.MORPH\_CLOSE, kernel)

# Applying static edge extraction

edges\_foreground = cv2.bilateralFilter(gray, 5, 55, 55)

edges\_filtered = cv2.Canny(edges\_foreground, 30, 60)

# Crop off the edges out of the moving area

cropped = (foreground // 255) \* edges\_filtered

return cropped

#cropped = Extractor()

# Auto Canny

# It doesn't work on OpenCV 3 as it does in OpenCV 4, can not be

# compiled on Rpi 3 with current versions.

# wide = cv2.Canny(gray\_gaussian, 10, 200)

# tight = cv2.Canny(gray\_gaussian, 225, 250)

# Stacking the images to print them together for comparison

#images = np.hstack((gray, edges, edges\_bilateral, edges\_high\_thresh))

# Image opening to see moving parts

# Drawing the center line <frame, start\_loc, end\_loc, colour, width>

LineDraw = cv2.line(gray,(320,0),( 320,480), (255,255,255) , 3)

#Convex Hull

CHull = edges\_bilateral #Giving CHull the image to compute

im2, contours, hierarchy = cv2.findContours(CHull, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# create hull array for convex hull points

hull = []

# create an empty black image

drawing = np.zeros((CHull.shape[0], CHull.shape[1], 3), np.uint8)

centers = np.zeros((CHull.shape[0], CHull.shape[1], 3), np.uint8)

# calculate points for each contour

for i in range(len(contours)):

# creating convex hull object for each contour

hull.append(cv2.convexHull(contours[i], False))

# draw object centers

cnt = hull[i]

M = cv2.moments(cnt) #Moment of the figure

if M["m00"] != 0:

Cx = int(M["m10"]/M["m00"])

Cy = int(M["m01"]/M["m00"])

cv2.line(centers,(Cx-3,Cy),(Cx+3,Cy),(0,255,0),3)#horizontal axis

cv2.line(centers,(Cx,Cy-3),(Cx,Cy+3),(0,255,0),3)#vertical axis

else:

# Unidentified exception

cX, cY = 0, 0

#cv2.imshow("Centers",centers)

# draw contours and hull points

for i in range(len(contours)):

color\_contours = (0, 0, 255) # red - color for contours

color = (255, 0, 0) # blue - color for convex hull

# draw ith contour

cv2.drawContours(drawing, contours, i, color\_contours, 1, 8, hierarchy)

# draw ith convex hull object

cv2.drawContours(drawing, hull, i, color, 1, 8)

# Overlaying

added\_stream = cv2.addWeighted(gray,0.2, edges\_bilateral,0.4,0) #camera feed with Canny

added\_calc = cv2.addWeighted(drawing,1,centers,1,0)

#Output Testing

# cv2.imshow('Output',drawing)

# cv2.imshow('Tester',SingleImg)

# cv2.imshow('Contours',drawing)

# cv2.imshow('Contours2',CHull)

# Display the resulting frame

# cv2.imshow('Base',gray) #Default cv2.COLOR\_BGR2GRAY check

# cv2.imshow('frame',images) #Canny test stack

# cv2.imshow('cropped', cropped) #Tesing Crop

# cv2.imshow("Edges", np.hstack([wide, tight])) #Auto Canny test stack

# cv2.imshow('Overlay',added\_stream)

cv2.imshow('Overlay2',added\_calc)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# When everything done, release the capture

cap.release()

cv2.destroyAllWindows()