МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЁТ

По лабораторной работе №3

по дисциплине «Аппаратное и программное обеспечение   
роботизированных систем»

«Программное обеспечение  
роботизированных систем»

ПРОВЕРИЛ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гай В.Е.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Абросимов К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тюрин К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Львутина Т.

**17-В-1**

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

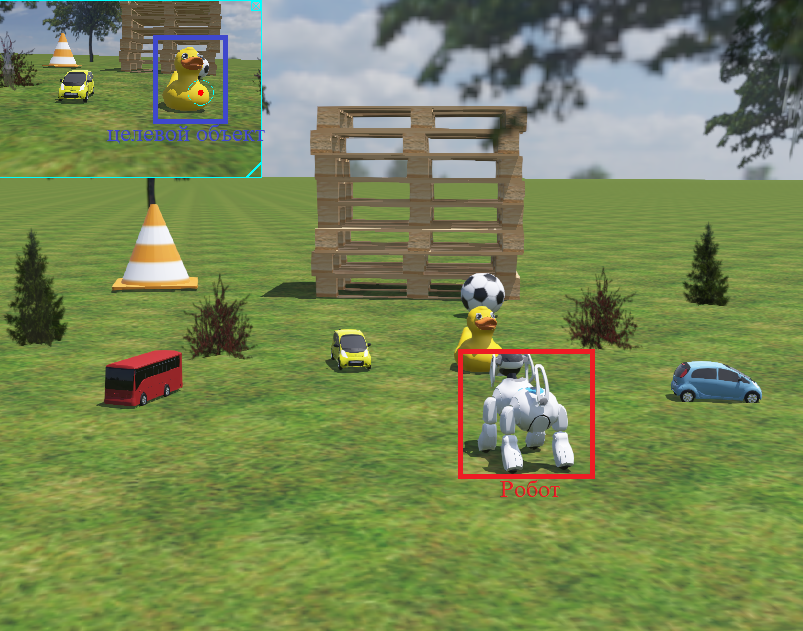
С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

2020

**Цель:** написать программу для робота по вариантам в среде симуляции Webots.

**Вариант:** **Отслеживание положения объекта.** Цель контроллера состоит в двух подзадачах: детектировать объект на изображении, повернуть и наклонить камеру так, чтобы целевой объект (уточка) был в центре изображения.



**Рекомендации:**

- применить морфологический фильтр (dilation и erosion) для удаления шума;

- использовать информацию из предыдущих фреймов для выбора наиболее подходящего участка изображения;

- использовании для поиска уточки методов: sift, surf.

- перевести изображения для поиска жёлтых объектов в другое цветовое пространство (например, HSL)

- использовать ограничения на геометрическую фигуру уточки;

- использовать алгоритмы фильтрации (фильтр Калмана, фильтр частиц) для предсказания положения уточки в следующем кадре.

- Улучшение качества слежения можно достигнуть если выбрать оптимальную скорость слежения.

**1. Выделение желтых контуров и код контроллера с комментариями.**

def cleanup():

"""Удаление файлов изображений с девайсов."""

# Игнорирование ошибок, если отсутствуют файлы.

try:

os.remove(deviceImagePath + '/display.jpg')

except OSError:

pass

try:

os.remove(deviceImagePath + '/camera.jpg')

except OSError:

pass

def sendDeviceImage(robot, device):

"""Отрисовка найденного объекта на дисплее."""

if device.getNodeType() == Node.DISPLAY:

deviceName = 'display'

fileName = deviceName + '.jpg'

device.imageSave(None, deviceImagePath + '/' + fileName)

elif device.getNodeType() == Node.CAMERA:

deviceName = 'camera'

fileName = deviceName + '.jpg'

device.saveImage(deviceImagePath + '/' + fileName, 80)

else:

return

with open(deviceImagePath + '/' + fileName, 'rb') as f:

fileString = f.read()

fileString64 = base64.b64encode(fileString).decode()

robot.wwiSendText("image[" + deviceName + "]:data:image/jpeg;base64," + fileString64)

f.close()

# Путь для сохранения изображений

deviceImagePath = os.getcwd()

try:

imageFile = open(deviceImagePath + "/image.jpg", 'w')

imageFile.close()

except IOError:

deviceImagePath = tempfile.gettempdir()

# Обращаемся к роботу

robot = Robot()

# Шаг контроллера

timestep = int(robot.getBasicTimeStep() \* 4)

# Получаем по имени камеру-мотор на роботе

panHeadMotor = robot.getMotor('PRM:/r1/c1/c2-Joint2:12')

tiltHeadMotor = robot.getMotor('PRM:/r1/c1/c2/c3-Joint2:13')

# Следующая камера-мотор не используется в контроллере

# tiltNeckMotor = robot.getMotor('PRM:/r1/c1-Joint2:11')

# Инициализируем моторы

panHeadMotor.setPosition(float('+inf'))

tiltHeadMotor.setPosition(float('+inf'))

# Установим скорость = 0

panHeadMotor.setVelocity(0.0)

tiltHeadMotor.setVelocity(0.0)

# Получаем доступ к камере робота, инициализируем и получаем характеристики камеры

camera = robot.getCamera('PRM:/r1/c1/c2/c3/i1-FbkImageSensor:F1')

camera.enable(timestep)

width = camera.getWidth()

height = camera.getHeight()

# Получаем дисплей робота

# Дисплей используется для отображения найденного целевого объекта

display = robot.getDisplay('display')

# Показать изображение камеры в фоновом режиме дисплея и установить цвет.

display.attachCamera(camera)

display.setColor(0xFF0000)

# Переменные, необходимые для отображения цели на дисплее.

targetPoint = []

targetRadius = 0

# Основной цикл.

while robot.step(timestep) != -1:

# Удаляет объекты дисплейный прицел, найденного объекта на прошлой итерации

if targetPoint:

# Стереть предыдущий дисплей нацел, выставив альфу = 0 и радиус.

display.setAlpha(0.0)

radius = targetRadius

if radius < 5:

# Минимальный размер красной точки

radius = 5

size = 2 \* radius + 1

display.fillRectangle(targetPoint[0] - radius,

targetPoint[1] - radius, size, size)

# Показать обзор камеры в окно робота.

# sendDeviceImage(robot, camera)

# Получить изображение с камеры.

rawString = camera.getImage()

# Создать маску для определения желтых пикселей.

index = 0

maskRGB = np.zeros([height, width], np.uint8)

for j in range(0, height):

for i in range(0, width):

# Формат изображения в пикселях.

b = rawString[index]

g = rawString[index + 1]

r = rawString[index + 2]

index += 4

# Пороги желтого цвета

if b < 50 and g > 180 and r > 180:

maskRGB[j][i] = True

# Найти контуры желтых пятен в маске.

contours = cv2.findContours(maskRGB.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[-2]

# Если не найден сгусток желтых пикселей, продолжай анализировать кадры с камеры.

if not contours:

continue

# Выбираем самый большой сгусток желтых пикселей.

blob = max(contours, key=cv2.contourArea)

# Вычислите минимальный окружающий круг и центроид данного сгустка.

((x, y), radius) = cv2.minEnclosingCircle(blob)

targetPoint = [int(x), int(y)]

targetRadius = int(radius)

# Показать обнаруженный объект на дисплее.

display.setAlpha(1.0)

if targetRadius > 0:

display.setColor(0x00FFFF)

display.drawOval(targetPoint[0], targetPoint[1], targetRadius, targetRadius)

display.setColor(0xFF0000)

display.fillOval(int(targetPoint[0]), int(targetPoint[1]), 5, 5)

# Показываем данные на дисплее

sendDeviceImage(robot, display)

# Качаем головой собаки-робота, чтобы цель была в центре изображения.

# Вычисление расстояние между целевым объектом и центром.

dx = targetPoint[0] - width / 2

dy = targetPoint[1] - height / 2

# Коэффициент подбираем эмпирически: 1.5.

panHeadMotor.setVelocity(-1.5 \* dx / width)

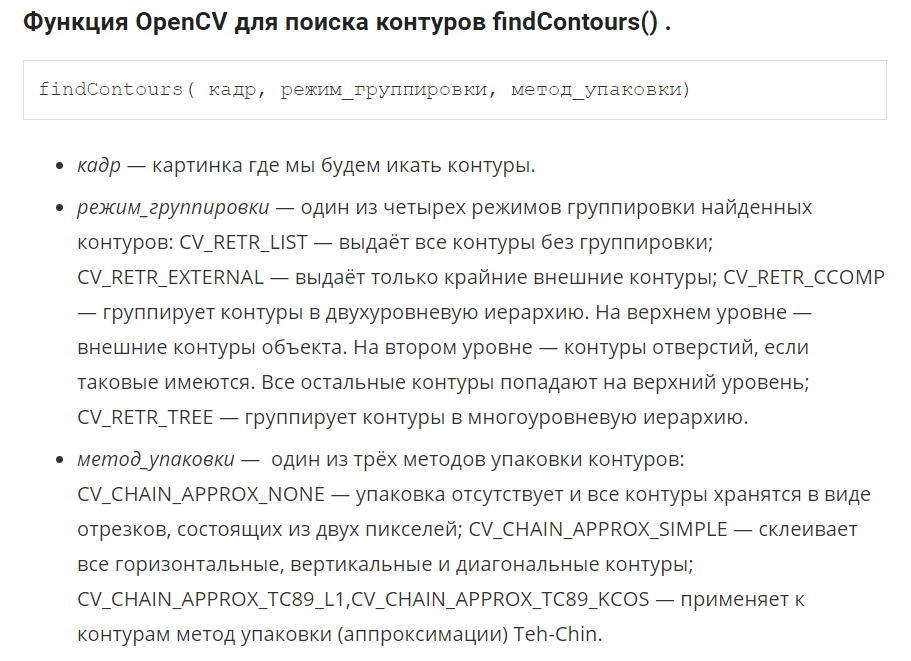
tiltHeadMotor.setVelocity(-1.5 \* dy / height)

# Код очистки.

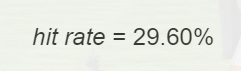
cleanup()

**Поиск контура в маске.**

cv2.findContours(maskRGB.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

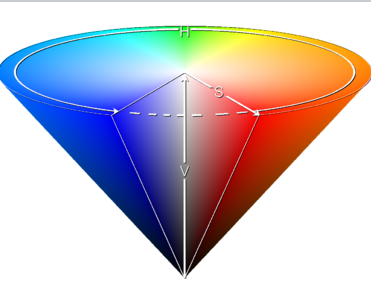
****

**КАЧЕСТВО:**

****

**2. Добавим переход из RGB пространства в HSV.**

**HSV** (англ. *Hue, Saturation, Value* — *тон*, *насыщенность*, *значение*) или **HSB** (англ. *Hue, Saturation, Brightness* — *тон*, *насыщенность*, *яркость*) — цветовая модель, в которой координатами цвета являются:

[https://upload.wikimedia.org/wikipedia/commons/thumb/a/ad/HueScale.svg/250px-HueScale.svg.png](https://commons.wikimedia.org/wiki/File:HueScale.svg?uselang=ru)

* **H**ue — цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0—360°, однако иногда приводится к диапазону 0—100 или 0—1.
* **S**aturation — насыщенность. Варьируется в пределах 0—100 или 0—1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому.
* **V**alue (значение цвета) или **B**rightness — яркость. Также задаётся в пределах 0—100 или 0—1.

Модель была создана Элви Рэем Смитом (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=Alvy_Ray_Smith&action=edit&redlink=1), одним из будущих со-основателей Pixar, в середине 1970-х. Она является нелинейным преобразованием модели RGB.



Дополнительная вставка:

MAX = max(r,g,b)

MIN = min(r,g,b)

if MAX - MIN == 0:

h\_hsv = 0

elif MAX == g:

h\_hsv = 60\*(b-r)/(MAX-MIN) + 120

elif MAX == b:

h\_hsv = 60\*(r-g)/(MAX-MIN) + 240

elif MAX == r and g >= b:

h\_hsv = 60\*(g-b)/(MAX-MIN) + 0

else:

h\_hsv = 60\*(g-b)/(MAX-MIN) + 360

s\_hsv = 0 if MAX == 0 else (1 - (MIN /MAX))

v\_hsv = MAX

index += 4

if 49 < h\_hsv < 55:

maskHSV[j][i] = True

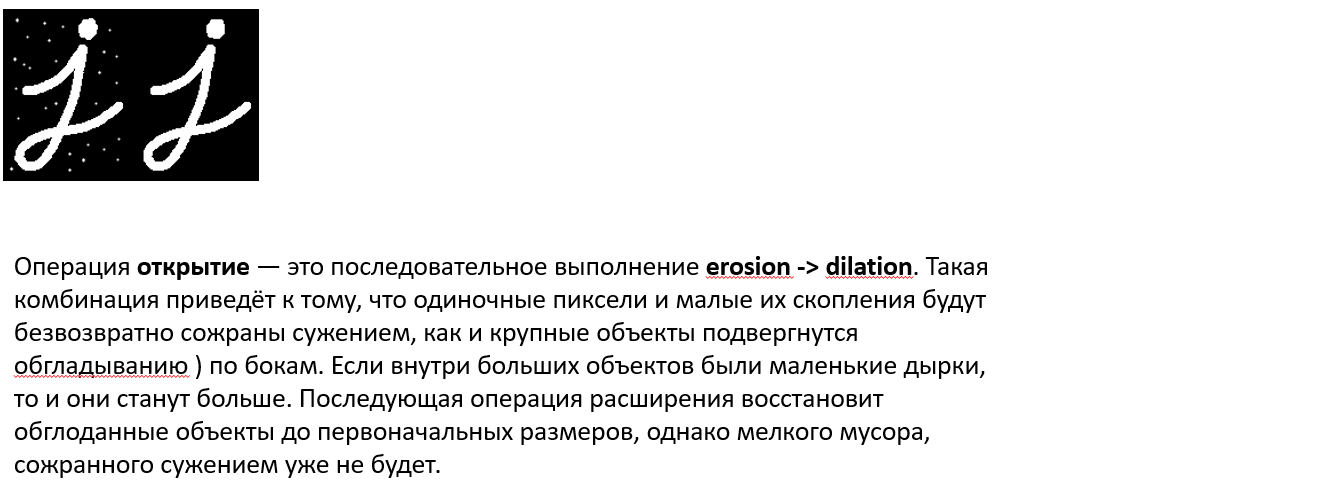
**КАЧЕСТВО:**

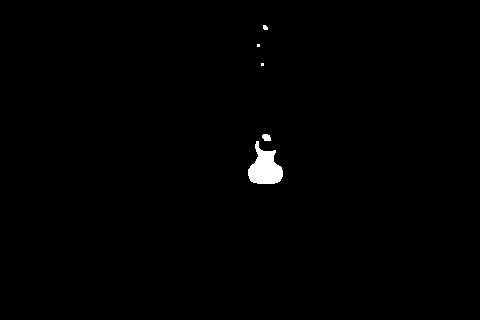
****

**3. Применим морфологические операторы: Дилатация, Эрозия -> Открытие.**



Эрозия и дилатация используются на бинарных изображениях.





Дополнительная вставка:

kernel = np.ones((3,3),np.uint8)

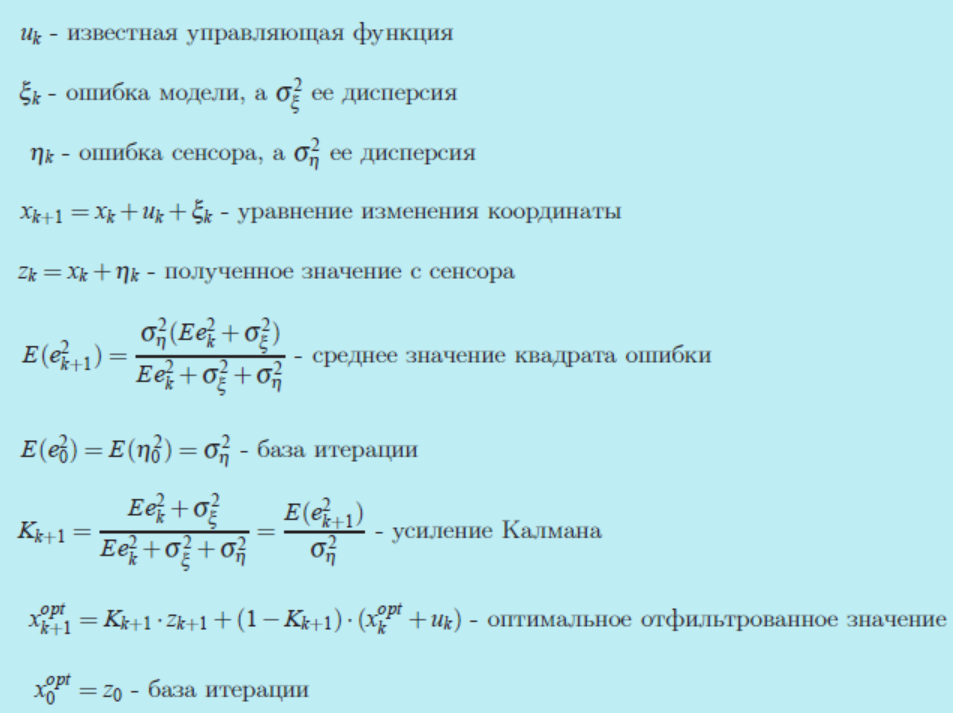
opening = cv2.morphologyEx(maskHSV, cv2.MORPH\_OPEN, kernel)

**КАЧЕСТВО (с шаблоном «открытия» 3х3):**

****

**4. Применение фильтра Калмана.**

Фильтр Калмана – эффективный рекурсивный фильтр, оценивающий вектор состояния динамической системы, используя ряд неполных и зашумленных измерений.

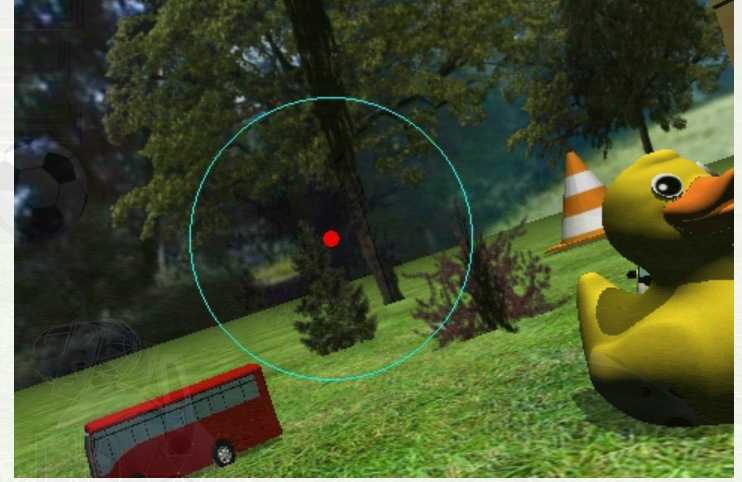


В нашем случае, мы рассматриваем две координаты: x и y. Следовательно, применяем отдельно фильтр Калмана для координаты x, и координаты y. Также, принимаем во внимание, что мы ничего особо о системе не знаем, следовательно, Uk – отсутствует.

К сожалению, фильтр Калмана достаточно плохо себя показал, изначально при настройке фильтра, цель была потеряна и слаба держалась самой цели, возможно, если применить другие изначальные настройки, сможем увеличить качество отслеживания.

**КАЧЕСТВО   
(с шаблоном «открытия» 5х5, СКО(измерений) = 3, СКО(точки) = 1, К0 = 0.9):**

****

****

Дополнительный код:

flag\_start\_kalman = True

sigmaMeasure = 3

sigmaNoise = 1

…

if flag\_start\_kalman:

flag\_start\_kalman = False

Ee\_x = [sigmaMeasure \*\* 2, 0]

Ee\_y = [sigmaMeasure \*\* 2, 0]

x\_opt = [targetPoint[0], targetPoint[0],0]

y\_opt = [targetPoint[1], targetPoint[1],0]

K\_x = K\_y = 0.9

else:

Ee\_x[1] = ((sigmaMeasure\*\*2)\*(Ee\_x[0]\*\*2+sigmaNoise\*\*2))/(Ee\_x[0]\*\*2+sigmaNoise\*\*2++sigmaMeasure\*\*2)

Ee\_y[1] = ((sigmaMeasure\*\*2)\*(Ee\_y[0]\*\*2+sigmaNoise\*\*2))/(Ee\_y[0]\*\*2+sigmaNoise\*\*2++sigmaMeasure\*\*2)

K\_x = Ee\_x[1]/(sigmaMeasure\*\*2)

K\_y = Ee\_y[1]/(sigmaMeasure\*\*2)

print(K\_x, K\_y)

x\_opt[2] = K\_x\*targetPoint[0] + (1-K\_x)\*(x\_opt[1]+0.5\*(x\_opt[1]-x\_opt[0]))

y\_opt[2] = K\_y\*targetPoint[1] + (1-K\_y)\*(y\_opt[1]+0.5\*(y\_opt[1]-y\_opt[0]))

targetPoint = [int(x\_opt[1]), int(y\_opt[1])]

…

Ee\_x[0] = Ee\_x[1]

Ee\_y[0] = Ee\_y[1]

x\_opt[0] = x\_opt[1]

x\_opt[1] = x\_opt[2]

y\_opt[0] = y\_opt[1]

y\_opt[1] = y\_opt[2]

**5. Применение упрощенного фильтра Калмана.**

Упрощенный фильтр Калмана – подразумевает, что коэффициент Калмана – не является итеративно-исчисляемым, а изначально заданным и неизменяющимся. Это позволит роботу не потерять цель изначально.

Также, все-таки введем Uk – как половина разности предыдущих измерений:   
Uk =0.5\*(Xk – Xk-1).

Эмпирически, был выбран коэффициент Калмана 0.875.

Дополнительный код:

flag\_start\_kalman = True

sigmaMeasure = 3

sigmaNoise = 1

…

if flag\_start\_kalman:

flag\_start\_kalman = False

x\_opt = [0,0,targetPoint[0], targetPoint[0],0]

y\_opt = [0,0,targetPoint[1], targetPoint[1],0]

K\_x = K\_y = 0.875

else:

x\_opt[4] = K\_x\*targetPoint[0] + (1-K\_x)\*(x\_opt[3]+0.5\*(x\_opt[3]-x\_opt[2]))

y\_opt[4] = K\_y\*targetPoint[1] + (1-K\_y)\*(y\_opt[3]+0.5\*(y\_opt[3]-y\_opt[2]))

targetPoint = [int(x\_opt[4]), int(y\_opt[4])]

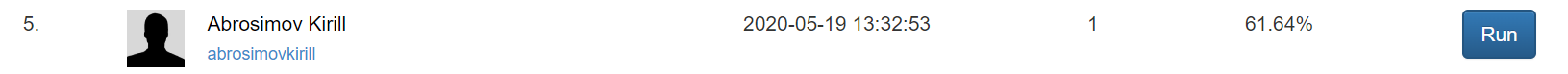
…

for ii in range(4):

x\_opt[ii] = x\_opt[ii+1]

y\_opt[ii] = y\_opt[ii+1]

**КАЧЕСТВО   
(с шаблоном «открытия» 5х5, СКО(измерений) = 3, СКО(точки) = 1, К = 0.875):**



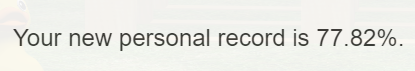
**6. Дополнительные условия + регулирование скорости.**

А) Несмотря на то, что оператор Открытия удаляет шум, он очень зависим от шаблона, и все оттенки желтого он не сможет вывести. Поэтому, когда наша уточка прячется за другими предметами робот начинает реагировать на любой такой шумок. Чтобы робот резко не поворачивался, было сделано условие, что, если координата очередной цели находится очень далеко, от найденной цели на прошлой итерации, поворот готовы робота – не изменяется, следовательно, робот продолжает поворачиваться и смотреть в направление, где, теоретически может находиться уточка.

Б) Также в ходе выполнения экспериментов, было выявлено, что на качество отслеживания влияет расположение уточки в кадре, то есть, даже если уточка детектировалась, но при этом она не находится в центре кадра, считается, что робот не справляется с задачей. Для этого был выявлен более оптимальный коэффициент поворота головы робота: 1.8 (было 1.5).

В результате, был полученный лучший результат за всю лабораторную работу: 78%.

**КАЧЕСТВО   
(с шаблоном «открытия» 5х5, СКО(измерений) = 3, СКО(точки) = 1, К = 0.875):**



**7. Применение двухфакторного фильтра Калмана с помощью openCV.**

****

kalman = cv2.KalmanFilter(4, 2)

kalman.measurementMatrix = np.array([[1, 0, 0, 0],

[0, 1, 0, 0]], np.float32)

kalman.transitionMatrix = np.array([[1, 0, 0, 0],

[0, 1, 0, 1],

[0, 0, 1, 0],

[0, 0, 0, 1]], np.float32)

kalman.processNoiseCov = np.array([[1, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]], np.float32) \* 0.03

…

kalman.correct(np.array([np.float32(x), np.float32(y)], np.float32))

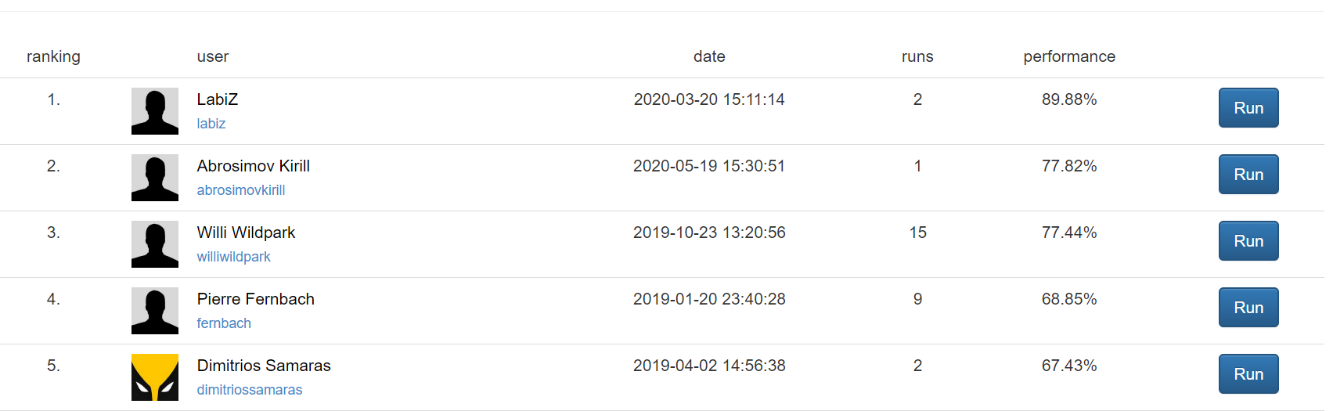
prediction = kalman.predict()

targetPoint = [int(prediction[0]), int(prediction[1])]

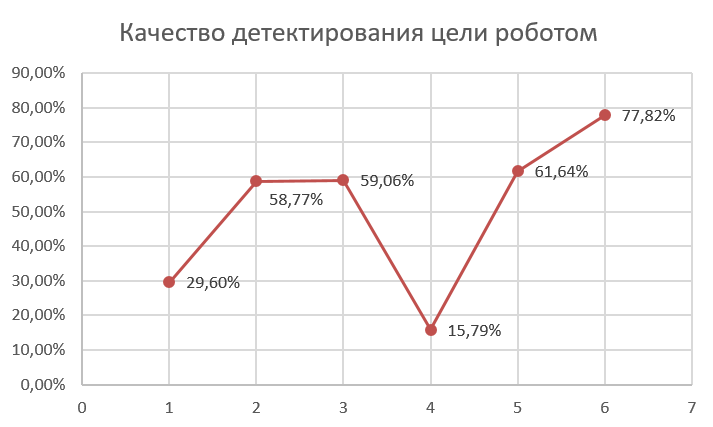
**Вывод:** в ходе выполнения лабораторной работы был реализован контроллер, позволяющий следить за целью на экране. Для этого были изучены:

* Цветовые пространства и переходы между ними.
* Фильтр Калмана и его применение.
* Морфологические операторы.
* Выделение контуров по маске.
* Предоставляемые функции библиотеки OpenCV.

Также в рамках данной задачи, в соревновании, было достигнуто 2 место на момент сдачи лабораторной работы (https://robotbenchmark.net/benchmark/visual\_tracking/).



Графики измерений:



1 – Определение желтого цвета в пространстве RGB, поиск контура.

2 - Определение желтого цвета в пространстве HSV, поиск контура.

3 - HSV, поиск контура, открытие 3х3.

4 - HSV, поиск контура, открытие 5х5, фильтр Калмана без Uk.

5 - HSV, поиск контура, открытие 5х5, фильтр Калмана упрощ.с Uk.

6 - HSV, поиск контура, открытие 5х5, фильтр Калмана упрощ.с Uk, доп.ограничения, регул.скорости.

**Итоговый код:**

from controller import Robot, Node

import base64

import os

import sys

def cleanup():

try:

os.remove('display.jpg')

except OSError:

pass

try:

os.remove('camera.jpg')

except OSError:

pass

def sendDeviceImage(robot, device):

if device.getNodeType() == Node.DISPLAY:

deviceName = 'display'

fileName = deviceName + '.jpg'

device.imageSave(None, fileName)

elif device.getNodeType() == Node.CAMERA:

deviceName = 'camera'

fileName = deviceName + '.jpg'

device.saveImage(fileName, 80)

else:

return

with open(fileName, 'rb') as f:

fileString = f.read()

fileString64 = base64.b64encode(fileString)

robot.wwiSendText("image[" + deviceName + "]:data:image/jpeg;base64," + str(fileString64))

f.close()

robot = Robot()

timestep = int(robot.getBasicTimeStep() \* 4)

panHeadMotor = robot.getMotor('PRM:/r1/c1/c2-Joint2:12')

tiltHeadMotor = robot.getMotor('PRM:/r1/c1/c2/c3-Joint2:13')

panHeadMotor.setPosition(float('+inf'))

tiltHeadMotor.setPosition(float('+inf'))

panHeadMotor.setVelocity(0.0)

tiltHeadMotor.setVelocity(0.0)

camera = robot.getCamera('PRM:/r1/c1/c2/c3/i1-FbkImageSensor:F1')

camera.enable(timestep)

width = camera.getWidth()

height = camera.getHeight()

display = robot.getDisplay('display')

display.attachCamera(camera)

display.setColor(0xFF0000)

targetPoint = []

targetRadius = 0

k = 0

flag\_start\_kalman = True

sigmaMeasure = 3

sigmaNoise = 1

err = 10

while robot.step(timestep) != -1:

if targetPoint:

# Erase the previous drawing by setting the pixels alpha value to 0 (transparent).

display.setAlpha(0.0)

radius = targetRadius

if radius < 5:

radius = 5

size = 2 \* radius + 1

display.fillRectangle(targetPoint[0] - radius,

targetPoint[1] - radius, size, size)

rawString = camera.getImage()

index = 0

flag\_go = True

maskHSV = np.zeros([height, width], np.uint8)

maskHSV\_morf = np.ones([height, width], np.uint8)

for j in range(0, height):

for i in range(0, width):

b = rawString[index]/255

g = rawString[index + 1]/255

r = rawString[index + 2]/255

MAX = max(r,g,b)

MIN = min(r,g,b)

if MAX - MIN == 0:

h\_hsv = 0

elif MAX == g:

h\_hsv = 60\*(b-r)/(MAX-MIN) + 120

elif MAX == b:

h\_hsv = 60\*(r-g)/(MAX-MIN) + 240

elif MAX == r and g >= b:

h\_hsv = 60\*(g-b)/(MAX-MIN) + 0

else:

h\_hsv = 60\*(g-b)/(MAX-MIN) + 360

s\_hsv = 0 if MAX == 0 else (1 - (MIN /MAX))

v\_hsv = MAX

index += 4

if 49 < h\_hsv < 55:

maskHSV[j][i] = True

kernel = np.ones((5,5),np.uint8)

opening = cv2.morphologyEx(maskHSV, cv2.MORPH\_OPEN, kernel)

contours = cv2.findContours(maskHSV.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[-2]

if not contours:

continue

blob = max(contours, key=cv2.contourArea)

((x, y), radius) = cv2.minEnclosingCircle(blob)

targetPoint = [int(x), int(y)]

targetRadius = int(radius)

if flag\_start\_kalman:

flag\_start\_kalman = False

x\_opt = [0,0,targetPoint[0], targetPoint[0],0]

y\_opt = [0,0,targetPoint[1], targetPoint[1],0]

K\_x = K\_y = 0.875

else:

if abs(targetPoint[0]-x\_opt[3])<60 or err!=0:

x\_opt[4] = K\_x\*targetPoint[0] + (1-K\_x)\*(x\_opt[3]+0.5\*(x\_opt[3]-x\_opt[2]))

if err!=0:

err -= 1

else:

flag\_go = False

x\_opt[4] = x\_opt[3]

#x\_opt[4] = (x\_opt[3]+0.9\*(sum([(x\_opt[1]-x\_opt[0]), (x\_opt[2]-x\_opt[1]),(x\_opt[3]-x\_opt[2])])/3))

if abs(targetPoint[1]-y\_opt[3])<40 or err!=0:

y\_opt[4] = K\_y\*targetPoint[1] + (1-K\_y)\*(y\_opt[3]+0.5\*(y\_opt[3]-y\_opt[2]))

if err!=0:

err -= 1

else:

#y\_opt[4] = (y\_opt[3]+0.9\*(sum([(y\_opt[1]-y\_opt[0]), (y\_opt[2]-y\_opt[1]),(y\_opt[3]-y\_opt[2])])/3))

flag\_go = False

y\_opt[4] = y\_opt[3]

targetPoint = [int(x\_opt[4]), int(y\_opt[4])]

display.setAlpha(1.0)

if targetRadius > 0:

display.setColor(0x00FFFF)

display.drawOval(targetPoint[0], targetPoint[1], targetRadius, targetRadius)

display.setColor(0xFF0000)

display.fillOval(int(targetPoint[0]), int(targetPoint[1]), 5, 5)

sendDeviceImage(robot, display)

if flag\_go:

dx = targetPoint[0] - width / 2

dy = targetPoint[1]- height / 2

panHeadMotor.setVelocity(-1.8 \* dx/width)

tiltHeadMotor.setVelocity(-1.8 \* dy/height)

for ii in range(4):

x\_opt[ii] = x\_opt[ii+1]

y\_opt[ii] = y\_opt[ii+1]

cleanup()