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

Лабораторная работа № 3

по дисциплине

Программное обеспечение роботизированных систем

РУКОВОДИТЕЛЬ:

Гай В.Е.

СТУДЕНТ:

Тарасов А. В.

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

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

Нижний Новгород 2022

Задача

отслеживание положения объекта: разработать контроллер, который отслеживает положение объекта в кадре и управляет камерой, чтобы следить за объектом более точно. Webots включает виртуальное пространство, в котором реализована сцена для решения данной задачи.

Листинг

*"""Sample Webots controller for the visual tracking benchmark."""*

from controller import Robot, Node

import base64

import os

import sys

if sys.version\_info.major > 2:

sys.exit("This controller program only works with Python 2.7.")

try:

import numpy as np

except ImportError:

sys.exit("Warning: 'numpy' module not found. Please check the Python modules installation instructions " +

"at 'https://www.cyberbotics.com/doc/guide/using-python'.")

try:

import cv2

except ImportError:

sys.exit("Warning: 'cv2' module not found. Please check the Python modules installation instructions " +

"at 'https://www.cyberbotics.com/doc/guide/using-python'.")

def cleanup():

*"""Remove device image files."""*

# Ignore errors if file doesn't exist.

try:

os.remove('display.jpg')

except OSError:

pass

try:

os.remove('camera.jpg')

except OSError:

pass

def sendDeviceImage(robot, device):

*"""Send the rendering device image to the robot window."""*

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()

# Get pointer to the robot.

robot = Robot()

# Set the controller time step based on the current world's time step.

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

# Get camera motors.

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

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

# Other camera motor not used in this controller.

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

# Initialize motors in order to use velocity control instead of position control.

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

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

# Set initial motors velocity.

panHeadMotor.setVelocity(0.0)

tiltHeadMotor.setVelocity(0.0)

# Get and enable the camera device.

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

camera.enable(timestep)

width = camera.getWidth()

height = camera.getHeight()

# Get the display device.

# The display can be used to visually show the tracked position.

display = robot.getDisplay('display')

# Show camera image in the display background.

display.attachCamera(camera)

display.setColor(0xFF0000)

# Variables needed to draw the target on the display.

targetPoint = []

targetRadius = 0

# Main loop: perform a simulation step until the simulation is over.

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

# Remove previously detected blob info from the display if needed.

if targetPoint:

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

display.setAlpha(0.0)

radius = targetRadius

if radius < 5:

# Minimum red dot size.

radius = 5

size = 2 \* radius + 1

display.fillRectangle(targetPoint[0] - radius,

targetPoint[1] - radius, size, size)

# Send the camera image to the robot window.

sendDeviceImage(robot, camera)

# Get camera image.

rawString = camera.getImage()

# Create mask for yellow pixels based on the camera image.

index = 0

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

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

for j in range(0, height):

for i in range(0, width):

# Camera image pixel format: BGRA.

#print(index)

b = ord(rawString[index])

g = ord(rawString[index + 1])

r = ord(rawString[index + 2])

index += 4

# Yellow color threshold.

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

maskRGB[j][i] = True

MAX = max(r, g, b)

MIN = min(r, g, b)

if MAX - MIN == 0:

hsvH = 0

elif MAX == g:

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

elif MAX == b:

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

elif MAX == r and g >= b:

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

else:

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

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

v\_hsv = MAX

if 49 < hsvH < 55:

maskHSV[j][i] = True

# Find blobs contours in the mask.

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

# Only proceed if at least one blob is found.

if len(contours) == 0:

continue

# Choose the largest blob.

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

# Compute the minimum enclosing circle and centroid of the blob.

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

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

targetRadius = int(radius)

# Show detected blob in the display: draw the circle and centroid.

display.setAlpha(1.0)

display.setColor(0x00FFFF)

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

display.setColor(0xFF0000)

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

# Send the display image to the robot window.

sendDeviceImage(robot, display)

# Move the head and camera in order to center the target object.

# Compute distance in pixels between the target point and the center.

dx = targetPoint[0] - width / 2

dy = targetPoint[1] - height / 2

# The speed factor 1.5 has been chosen empirically.

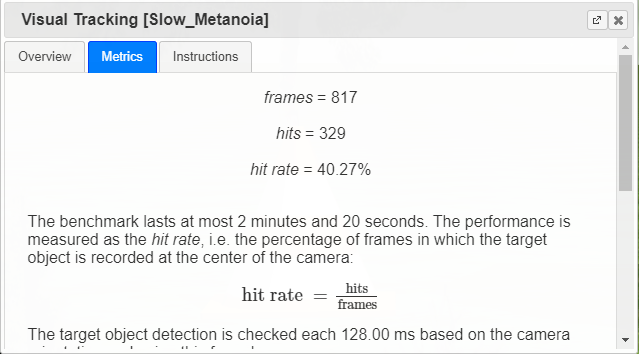
panHeadMotor.setVelocity(-1.0 \* dx / width)

tiltHeadMotor.setVelocity(-1.0 \* dy / height)

# Cleanup code.

cleanup()

Результат:



Для прохождения бенчмарка был изменён базовый контроллер. Маска выделения жёлтых пикселей была изменена с RGB на HSV. Также была незначительно скорректирована скорость поворота головы робота. Низкий процент попаданий в основном объясняется ложным определением шама как объекта, при полном пропадании объекта с камеры робота.