Apéndice A

Comandos UnrealCV

CHAPTER 11

Command System

Unreal Engine 4 has some built-in commands to help game development. These commands can be typed into a built-in console. Using these commands, a developer can profile the game performance and view debug information. To invoke the built-in console of a game, type the ‘ key (the key above tab).

UnrealCV provides commands useful for computer vision researchers. What is more, these commands can be used by an external program. A built-in command can also be used using the special command `vrun`.

11.1 Command cheatsheet

See this ipython notebook to see an imcomplete demo of available commands.

11.2 1. Camera operation

See `Source/UnrealCV/Private/Commands/CameraHandler.h(.cpp)` for more details.

vget /camera/[id]/location (v0.2) Get camera location [x, y, z]

vget /camera/[id]/rotation (v0.2) Get camera rotation [pitch, yaw, roll]

vset /camera/[id]/location [x] [y] [z] (v0.2) Set camera location [x, y, z]

vset /camera/[id]/rotation [pitch] [yaw] [roll] (v0.2) Set camera rotation [pitch, yaw, roll]

vget /camera/[id]/[viewmode] (v0.2) Get [viewmode] from the [id] camera, for example: vget /camera/0/depth

vget /camera/[id]/[viewmode] [filename] (v0.2) Same as the above, with an extra parameter for filename

> **filename** Filename is where the file will be stored.
>
> **example** `vget /camera/0/lit lit.png`

vget /camera/[id]/[viewmode] [format] (v0.3.7) Support binary data format

format If only file format is specified, the binary data will be returned through socket instead of being saved as a file.

example `vget /camera/0/lit png`

vget /camera/[id]/object_mask (v0.2) The object mask is captured by first switching the viewmode to object_mask mode, then take a screenshot

vset /viewmode [viewmode] (v0.2) Set ViewMode to (lit, normal, depth, object_mask)

vget /viewmode (v0.2) Get current ViewMode

vget /camera/[id]/pose (v0.3.10) Get camera location [x, y, z] and rotation [pitch, yaw, roll]

vset /camera/[id]/pose [x] [y] [z] [pitch] [yaw] [roll] (v0.3.10) Teleport camera to location [x, y, z] and rotation [pitch, yaw, roll]

vget /camera/[uint]/horizontal_fieldofview (v0.3.10) Get camera horizontal field of view

vset /camera/[uint]/horizontal_fieldofview [FOV] (v0.3.10) Set camera horizontal field of view

vget /camera/[uint]/vis_depth npy (v0.3.10)

vget /camera/[uint]/plane_depth npy (v0.3.10)

11.3 2. Object interaction

See *Source/UnrealCV/Private/Commands/ObjectHandler.h(.cpp)* for more details

vget /objects (v0.2) Get the name of all objects

vget /object/[obj_name]/color (v0.2) Get the labeling color of an object (used in object instance mask)

vget /object/[obj_name]/location Get object location [x, y, z]

vget /object/[obj_name]/rotation Get object rotation [pitch, yaw, roll]

vset /object/[obj_name]/location [x] [y] [z] Set object location [x, y, z]

vset /object/[obj_name]/rotation [pitch] [yaw] [roll] Set object rotation [pitch, yaw, roll]

vset /object/[obj_name]/color [r] [g] [b] (v0.2) Set the labeling color of an object

vset /object/[str]/show (v0.3.10) Show object

vset /object/[str]/hide (v0.3.10) Hide object

11.4 3. Plugin commands

See *Source/UnrealCV/Private/Commands/PluginHandler.h(.cpp)* for more details.

vget /unrealcv/status (v0.2) Get the status of UnrealCV plugin

vget /unrealcv/help (v0.2) List all available commands and their help message

11.5 4. Action commands

See `Source/UnrealCV/Private/Commands/ActionHandler.h(.cpp)`

vset /action/keyboard [key_name] [delta] (v0.3.6) Valid key_name can be found in here

vset /action/game/pause (v0.3.10) Pause the game

vset /action/game/level [level_name] (v0.3.10) Open a new level

vset /action/input/enable (v0.3.10) Enable input

vset /action/input/disable (v0.3.10) Disable input

vset /action/eyes_distance [eye_distance] (v0.3.10) Set the eye distance between left eye and right eye (camera 1). This command might be marked as deprecated when we finish multiple camera support.

11.6 Run UE4 built-in commands

vrun [cmd] (v0.3) This is a special command used to execute Unreal Engine built-in commands. UE4 provides some built-in commands for development and debug. They are not very well documented, but very useful.

A few examples are:

- `stat FPS` - show current frame rate
- `shot` - take a screenshot
- `show Material` - toggle the display of Material

These commands can be executed in the UE4 console. If you want to use them in UnrealCV, you can prefix these commands with *vrun stat FPS*.

11.7 Run Blueprint commands

vexec [cmd] TODO

Apéndice B

Implementación Panorama no central en Python

```
from unrealcv import client
from PIL import Image
import numpy as np
import math
import functions as f

#-----------------------------------------------------------------------------
#No central Panorama image set-up
final_w = 1024
final_h = 512
mode_list = ["lit","object_mask","depth"]
radius = 100
threshold = 50.0
#-----------------------------------------------------------------------------
loc_file = open('camera_loc.txt', 'r')
location = loc_file.read().split('\n')
start = input('Introduce starting index: ')
if start == None:
  start = 1

#get images on UnrealCV
def get_front(angle,mode):
  client.request('vset /camera/0/rotation 0 {} 0'.format(angle))
  client.request('vget /camera/0/{} {}_front.png'.format(mode,mode))
  if mode == 'lit':
  l_img = client.request('vget /camera/0/{} npy'.format(mode))
  f.lit_image2(l_img,"{}_".format(mode),"front")
  elif mode == 'object_mask':
  img = Image.open("{}_front.png".format(mode))
```

```
    width,height = img.size
    img.crop((-2+width/2,0,2+width/2,height))
      .save("{}_front.png".format(mode))
    elif mode == "depth":
    d_img = client.request('vget /camera/0/{} npy'.format(mode))
    f.depth_image2(d_img,"no-central/depth/depth_","front",threshold)

def get_up(angle,mode):
    client.request('vset /camera/0/rotation 90 {} 0'.format(angle))
    client.request('vget /camera/0/{} {}_up.png'.format(mode,mode))
    if mode == 'lit':
    l_img = client.request('vget /camera/0/{} npy'.format(mode))
    f.lit_image2(l_img,"{}_".format(mode),"up")
    elif mode == 'object_mask':
    img = Image.open("{}_up.png".format(mode))
    width,height = img.size
    img.crop((-2+width/2,0,2+width/2,height))
      .save("{}_up.png".format(mode))
    elif mode == "depth":
    d_img = client.request('vget /camera/0/{} npy'.format(mode))
    f.depth_image2(d_img,"no-central/depth/depth_","up",threshold)

def get_down(angle,mode):
    client.request('vset /camera/0/rotation -90 {} 0'.format(angle))
    client.request('vget /camera/0/{} {}_down.png'.format(mode,mode))
    if mode == 'lit':
    l_img = client.request('vget /camera/0/{} npy'.format(mode))
    f.lit_image2(l_img,"{}_".format(mode),"down")
    elif mode == 'object_mask':
    img = Image.open("{}_down.png".format(mode))
    width,height = img.size
    img.crop((-2+width/2,0,2+width/2,height))
      .save("{}_down.png".format(mode))
    elif mode == "depth":
    d_img = client.request('vget /camera/0/{} npy'.format(mode))
    f.depth_image2(d_img,"no-central/depth/depth_","down",threshold)

#Geometric parameters
Rx_pos = np.matrix('0 1 0;0 0 -1;1 0 0')
Rz_pos = np.matrix('0 -1 0;1 0 0;0 0 1')
nz_pos = np.matrix('0;0;1')
Rz_neg = np.matrix('0 -1 0;-1 0 0;0 0 -1')
nz_neg = np.matrix('0;0;-1')

#Change of coordinate system
```

```
def car2sph(x, y, n, m):
  theta = (2*x/float(n) - 1.0)*math.pi
  phi = (1/2.0 - y/float(m))*math.pi
  x_og = math.cos(phi)*math.cos(theta)
  y_og = math.cos(phi)*math.sin(theta)
  z_og = math.sin(phi)
  vec = np.matrix('{};{};{}'.format(x_og,y_og,z_og))
  return vec,phi

#Main program
client.connect()
if not client.isconnected():
  print('UnrealCV server is not running')
else:
  for mode in mode_list:
  final = Image.new("RGBA",(final_w,final_h),"white")
  pixels = final.load()
  client.request('vset /camera/0/horizontal_fieldofview 120')
  if mode == "depth":
    depth = np.empty([final_w,final_h],np.float32)
  f.folders('no-central',mode,0)
  #Pixel mapping
  for pos in range(len(location)-1):
    center_loc = location[pos].split(' ')
    for x in range(final_w):
    theta = (2*x/float(final_w) - 1.0)*math.pi
    #Camera positioning
    x_cam = int(center_loc[0]) + radius * np.cos(theta)
    y_cam = int(center_loc[1]) + radius * np.sin(theta)
    z_cam = int(center_loc[2])
    client.request('vset /camera/0/location {} {} {}'
        .format(x_cam,y_cam,z_cam))
    angle = np.rad2deg(theta)
    #Take images
    get_up(angle,mode)
    get_front(angle,mode)
    get_down(angle,mode)
    imagenx = Image.open('{}_front.png'.format(mode))
    imagenz_pos = Image.open('{}_up.png'.format(mode))
    imagenz_neg = Image.open('{}_down.png'.format(mode))
    im_w, im_h = imagenx.size
    im_w -=1
    im_h -=1
    #Camera parameters
    FOV = math.pi/2.0
```

```
fx = (im_w/2.0)/math.tan(FOV/2.0)
fy = (im_h/2.0)/math.tan(FOV/2.0)
K = np.matrix('{} 0 {}; 0 {} {}; 0 0 1'
    .format(fx,im_w/2,fy,im_h/2))
#Geometric parameters
cos = np.cos(theta)
sin = np.sin(theta)
R = np.matrix('{} {} 0;{} {} 0;0 0 1'
    .format(cos,sin,-sin,cos))
nx = np.matrix('{};{};0.0'.format(cos,sin))
Rx = Rx_pos * R
R_pos = Rz_pos * R
R_neg = Rz_neg * R
for y in range(final_h):
  vec,phi = car2sph(x, y, final_w, final_h)
  if 0.0 <= phi <= math.pi/2.0:
  imagen,R,n = f.choose_image2(vec, imagenx, Rx, nx,
          imagenz_pos, R_pos, nz_pos)
  elif -math.pi/2.0 <= phi < 0.0:
  imagen,R,n = f.choose_image2(vec, imagenx, Rx, nx,
          imagenz_neg, R_neg, nz_neg)

  #No central Panorama image build
  p_x, p_y = f.get_pixel(vec, R, K)
  color = imagen.getpixel((p_x, p_y))
  pixels[x, y] = color
    if mode == "depth":
  depth[x,y] = f.depth_file2(n,p_x,p_y,nx)
final.save("no-central/panorama_no-central.png")
```

Apéndice C

Implementación Catadióptricos no centrales en Python

```
from unrealcv import client
from PIL import Image
import numpy as np
import math
import functions as f
import os

#Interface
auto = raw_input('Use file Set-up (Yes) or enter new parameters(No): ')

if auto == 'Yes':
#-------------------------------------------------------------------------------
  #Catadioptric image set-up
  final_w = 1024
  final_h = 1024
  mode_list = ["lit"]
  mirror = 'spheric'
  R_s = 0.3
  tau = np.radians(55)
  Z_m = 0.2
  view_dir = 'z'
  view_sign = 'pos'
  data_set = False
#-------------------------------------------------------------------------------
else:
  final_w = input('Enter width: ')
  final_h = input('Enter height: ')
  mode_list = [raw_input('Enter mode: lit/object_mask/depth ')]
  mirror = raw_input('Enter mirror conic/spheric: ')
```

```
  tau,R_s = 0,0
  if mirror == 'spheric':
    R_s = input('Enter mirror radius: ')
  else:
    tau = np.radians(input('Enter cone angle: '))
  Z_m = input('Enter distance between camera and mirror: ')
  view_dir = raw_input('Enter view direction axis: ')
  view_sign = raw_input('Enter axis sign: pos/neg ')
  data_set = input('Data set? True/False: ')
#-------------------------------------------------------------------------------

loc_file = open('camera_loc.txt','r')
rot_file = open('camera_rot.txt', 'r')
rotation = rot_file.read().split(' ')
location = loc_file.read().split('\n')
x_past = 10000
y_past = 10000
Zr_past = 1000

#Conic catadioptric
R_c = Z_m*np.sin(2*tau)
Z_c = Z_m*(1-np.cos(2*tau))
Z_s = Z_m + R_s

#Geometric parameters
Rx_pos = np.matrix('0 -1 0;0 0 -1;1 0 0')
nx_pos = np.matrix('1;0;0')
Rx_neg = np.matrix('0 1 0;0 0 -1;-1 0 0')
nx_neg = np.matrix('-1;0;0')
Ry_pos = np.matrix('1 0 0;0 0 -1;0 1 0')
ny_pos = np.matrix('0;1;0')
Ry_neg = np.matrix('-1 0 0;0 0 -1;0 -1 0')
ny_neg = np.matrix('0;-1;0')
Rz_pos = np.matrix('0 -1 0;1 0 0;0 0 1')
nz_pos = np.matrix('0;0;1')
Rz_neg = np.matrix('0 -1 0;-1 0 0;0 0 -1')
nz_neg = np.matrix('0;0;-1')

#Camera parameters
R_view = f.camera_direction(view_dir,view_sign)
FOV = math.pi/2.0
FOV_cata = math.pi/2.0
f_cata = (final_w/2.0)/math.tan(FOV_cata/2.0)

#Functions
```

```
def get_param(mirror, Z_c, R_c, tau, r, Z_s, R_s):
  if mirror == 'conic':
  cot_phi = (1+r*np.tan(2*tau))/(np.tan(2*tau)-r)
  Z_r = Z_c + R_c*cot_phi
  elif mirror == 'spheric':
  Z_rel = Z_s/R_s
  rho = np.sqrt(r**2+1)
  gamma = (-r**2*Z_rel**2 + rho**2)#*Z_rel**2
  if gamma < 0:
    cot_phi = 1/r
    Z_r = 0
  else:
    xi = np.sqrt(gamma*Z_rel**2)
    d = 2*r**2*Z_rel**4 - 2*xi*Z_rel**2 - 3*rho**2*Z_rel**2
       + rho**2
    eps = (-r**2 + 1)*Z_rel**2 + 2*xi + rho**2
    dseta = 2*r**2*Z_rel**4 - 2*xi*(-r**2*Z_rel**2 + rho**2)
         - rho**2*(1 + Z_rel**2)
    cot_phi = -dseta/(d*r)
    Z_r = (d+eps)*Z_s/d
  else:
  print('ERROR: Wrong mirror input')
  return cot_phi,Z_r

def get_images(Zr,mirror,cam,direction,sign):
  if sign == 'neg':
  s = -100
  else:
  s = 100
  if direction == 'x':
  delta = np.matrix('{},0,0'.format(s*Zr))
  elif direction == 'y':
  delta = np.matrix('0,{},0'.format(s*Zr))
  else:
  delta = np.matrix('0,0,{}'.format(s*Zr))
  client.request('vset /camera/0/location {} {} {}'
    .format(int(cam[0])+delta[0,0],int(cam[1])+delta[0,1],
        int(cam[2])+delta[0,2]))
  for index in range(len(rotation)//3):
  angle = rotation[index*3:index*3+3]
  client.request('vset /camera/0/rotation {} {} {}'
      .format(int(angle[0]),int(angle[1]),int(angle[2])))
  client.request('vget /camera/0/lit {}/lit{}.png'.format(mirror,index))
  img = Image.open('{}/lit{}.png'.format(mirror,index))
  img.crop((187,187,699,699)).save('{}/lit{}.png'.format(mirror,index))
```

```
#Rotates to camera direction
def rot2cam(vec, R):
  e = 1e-6
  R_cam = np.matrix('0 -1 0;-1 0 0;0 0 1')
  vec_cam = R * R_cam * vec
  phi = np.arccos(vec_cam[2,0])
  if -e < vec_cam[1,0] < e:
  if vec_cam[0,0] >= 0:
    theta = 0.0
  else:
    theta = -math.pi
  elif vec_cam[1,0] > 0:
  theta = np.arccos(vec_cam[0,0]/np.sin(phi))
  elif vec_cam[1,0] < 0:
  theta = -np.arccos(vec_cam[0,0]/np.sin(phi))
  return phi, theta, vec_cam

#main program
client.connect()
if not client.isconnected():
  print('ERROR: Client is not connected')
else:
  #Camera images - skybox
  for pos in range(len(location)-1):
  camera_loc = location[pos].split(' ')
  for mode in mode_list:
    final = Image.new("RGBA",(final_w,final_h),"black")
    pixels = final.load()
    if mode == "depth":
    depth = np.empty([final_w,final_h],np.float32)
    f.folders('catadioptric',mode,0)
    #Pixel mapping
    for r in np.arange(1,final_w/2,1):
    if r < final_w/4:
      step = 0.1
    else:
      step = 0.05
    r_hat = r/f_cata
    cot_phi, Z_r = get_param(mirror, Z_c, R_c, tau, r_hat, Z_s, R_s)
    phi_cata = math.pi/2.0 - np.arctan(cot_phi)
    get_images(Z_r, mirror, camera_loc, view_dir, view_sign)
    imagenx_pos = Image.open('{}/lit0.png'.format(mirror))
    imagenx_neg = Image.open('{}/lit1.png'.format(mirror))
    imageny_pos = Image.open('{}/lit2.png'.format(mirror))
```

```
imageny_neg = Image.open('{}/lit3.png'.format(mirror))
imagenz_pos = Image.open('{}/lit4.png'.format(mirror))
imagenz_neg = Image.open('{}/lit5.png'.format(mirror))
im_w, im_h = imagenx_pos.size
im_w -= 1
im_h -= 1
fx = (im_w/2.0)/math.tan(FOV/2.0)
fy = (im_h/2.0)/math.tan(FOV/2.0)
K = np.matrix('{} 0 {}; 0 {} {}; 0 0 1.0'
    .format(fx,im_w/2,fy,im_h/2))
for angle in np.arange(0,360,step):
  theta_cata = np.radians(angle)
  x = int(math.floor(final_w/2 + r*np.cos(theta_cata)))
  y = int(math.floor(final_h/2 - r*np.sin(theta_cata)))
  if x==x_past and y==y_past:
    continue
  else:
  x_past = x
  y_past = y
  z_vec = np.cos(phi_cata)
  y_vec = np.sin(phi_cata)*np.sin(theta_cata)
  x_vec = np.sin(phi_cata)*np.cos(theta_cata)
  vec_cone = np.matrix('{};{};{}'.format(x_vec,y_vec,z_vec))
  phi,theta,vec = rot2cam(vec_cone,R_view)
  if 0.0 <= phi < math.pi/2.0:
    if -math.pi <= theta < -math.pi/2.0:
    imagen,R,n = f.choose_image(vec, imagenx_neg, Rx_neg, nx_neg,
          imageny_neg, Ry_neg, ny_neg, imagenz_pos, Rz_pos, nz_pos)
    elif -math.pi/2.0 <= theta < 0.0:
    imagen,R,n = f.choose_image(vec, imagenx_pos, Rx_pos, nx_pos,
          imageny_neg, Ry_neg, ny_neg, imagenz_pos, Rz_pos, nz_pos)
    elif 0.0 <= theta < math.pi/2.0:
    imagen,R,n = f.choose_image(vec, imagenx_pos, Rx_pos, nx_pos,
        imageny_pos, Ry_pos, ny_pos, imagenz_pos, Rz_pos, nz_pos)
    elif math.pi/2.0 <= theta <= math.pi:
    imagen,R,n = f.choose_image(vec, imagenx_neg, Rx_neg, nx_neg,
        imageny_pos, Ry_pos, ny_pos, imagenz_pos, Rz_pos, nz_pos)
  elif math.pi/2.0 <= phi <= math.pi:
    if -math.pi <= theta < -math.pi/2.0:
    imagen,R,n = f.choose_image(vec, imagenx_neg, Rx_neg, nx_neg,
        imageny_neg, Ry_neg, ny_neg, imagenz_neg, Rz_neg, nz_neg)
    elif -math.pi/2.0 <= theta < 0.0:
    imagen,R,n = f.choose_image(vec, imagenx_pos, Rx_pos, nx_pos,
        imageny_neg, Ry_neg, ny_neg, imagenz_neg, Rz_neg, nz_neg)
    elif 0.0 <= theta < math.pi/2.0:
```

```
        imagen,R,n = f.choose_image(vec, imagenx_pos, Rx_pos, nx_pos,
            imageny_pos, Ry_pos, ny_pos, imagenz_neg, Rz_neg, nz_neg)
        elif math.pi/2.0 <= theta <= math.pi:
        imagen,R,n = f.choose_image(vec, imagenx_neg, Rx_neg, nx_neg,
            imageny_pos, Ry_pos, ny_pos, imagenz_neg, Rz_neg, nz_neg)

    #Catadioptric image build
      p_x, p_y = f.get_pixel(vec, R, K)
      color = imagen.getpixel((p_x, p_y))
      if mode == "depth":
        depth[x,y] = f.depth_file(n,p_y,p_x,loc)
      pixels[x, y] = color
    final.save("{}/catadioptric -{}.png".format(mirror,mirror))
```

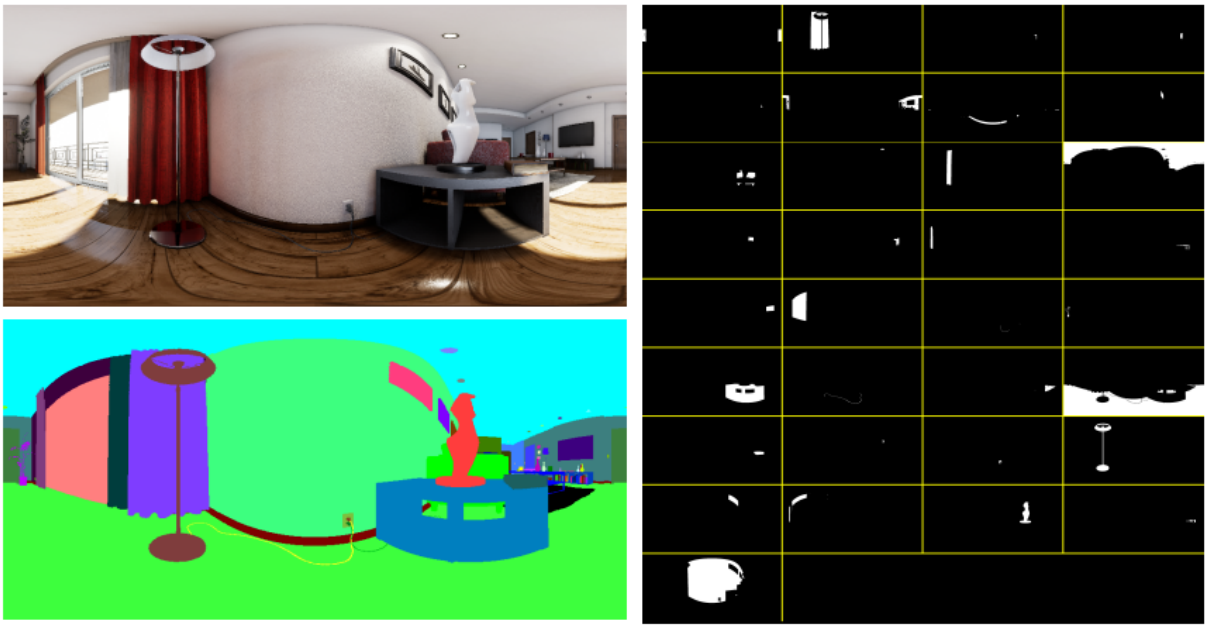
Apéndice D

Máscaras binarias

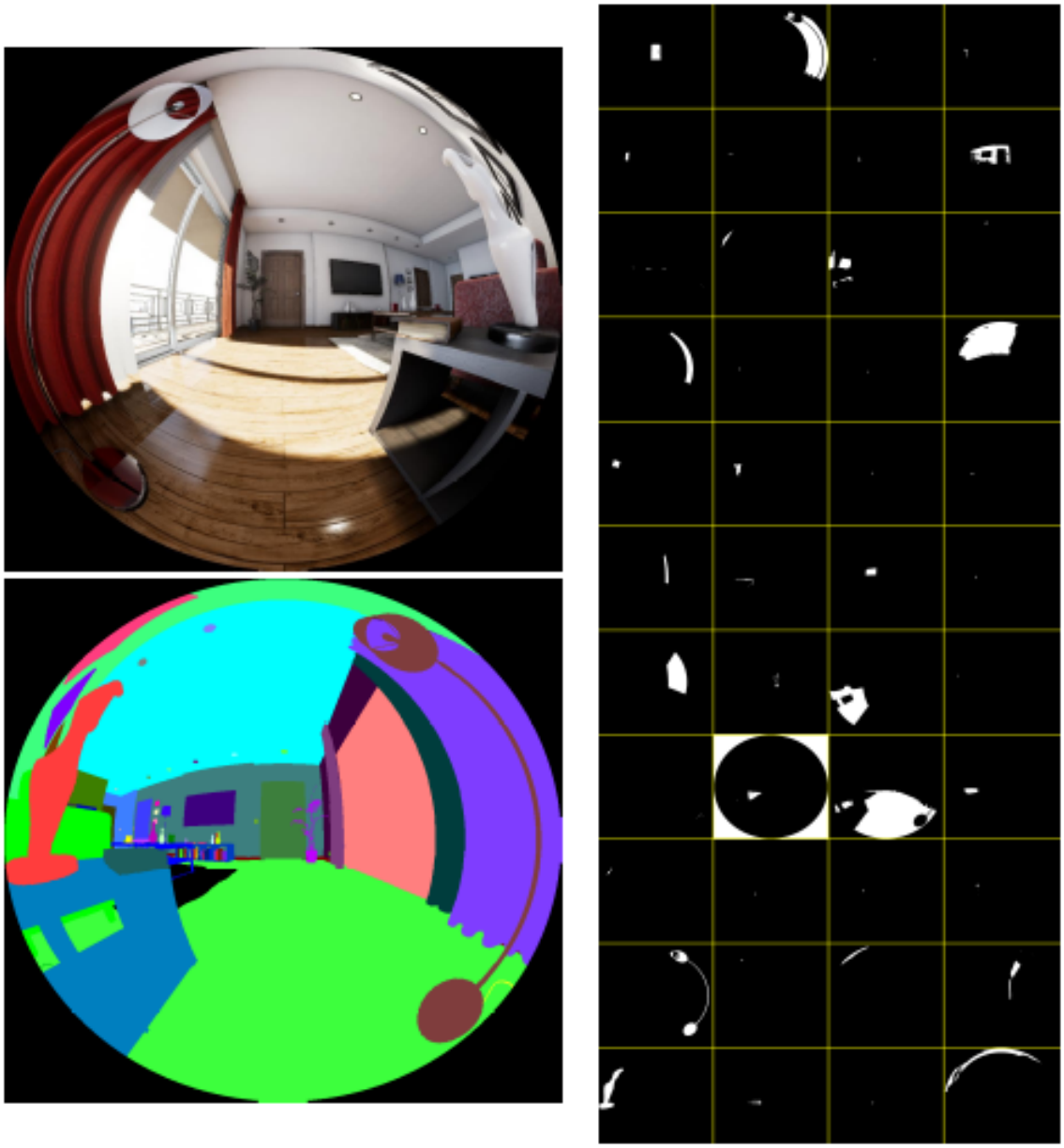
Este anexo muestra un ejemplo de máscaras binarias para reconocimiento de objetos. Se muestran ejemplos para distintos modelos proyectivos de cámaras omnidireccionales.

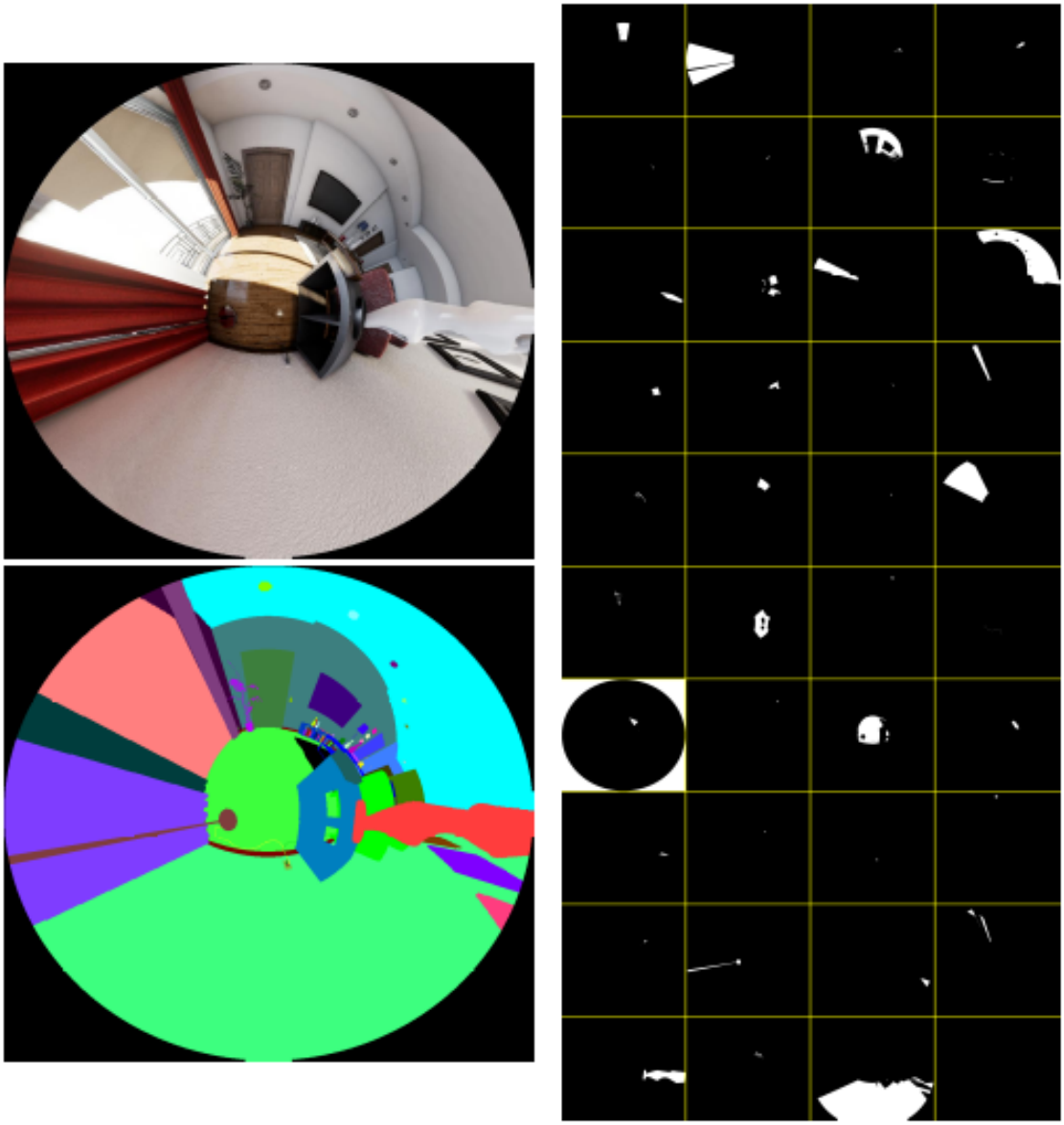
El formato de estas máscaras binarias se ha hecho de forma que coincida con el de [28]. Esto se ha hecho para poder evaluar la funcionalidad de estas máscaras y comprobar si se podría entrenar este algoritmo de visión por computador. No obstante, debido a lo reciente de este trabajo, no se ha podido realizar todavía una evaluación.

Como ejemplo de las máscaras binarias, primero se va a mostrar para imágenes panorámicas equirectangulares, siguiendo el trabajo de [28]:



Además del modelo equirectangular, este tipo de máscaras binarias se puede aplicar a cualquier modelo de cámara implementado en el simulador. Como ejemplo, a continuación se muestran máscaras binarias para imágenes de ojo de pez y catadióptricas.





Apéndice E

Simulador

En este anexo se muestra el repositorio de GitHub donde se subirá próximamente el código de este simulador, *https://github.com/Sbrunoberenguel/ImageSimulator*. Aunque en los anexos B y C se ha introducido el código para cámaras no centrales, no se ha introducido todo el código desarrollado en este proyecto. Se han presentado algunos extractos de código en la sección 4.2, pero el cógido completo parece muy extenso para presentarlo en este proyecto. Próximamente, se procederá a completar el repositorio y se podrá acceder al código de este simulador desde dicho repositorio.

Central and No-Central camera simulator

Python 2.7 implementation for a camera simulator using UnrealCV as virtual scenario.

Introduction

This repository contains the python scripts to obtain synthetic photo-realistic images from a list of camera models.

In this simulator, we include two types of cameras: central and no-central cameras.

For the central cameras, we have implemented:

- Equirectangular
- Cylindric
- Scaramuzza model
- Catadioptric cameras:
    - Para-cataadioptric
    - Hyper-catadioptric
    - Planar-catadioptric
- Fish eye cameras:
    - Equiangular

- Stereographic
- Equisolid angle
- Orthogonal

For the no central cameras, we have implemented:

- Panoramic
- Catadioptric cameras:
  - Spheric mirror
  - Conic mirror

Use instructions

Install requirements

This code has been compiled and tested in Ubuntu 18.04 LTS using:

- Python 2.7

Python libraries

To use this image simulator, first you have to make sure that you have isntalled the following python libraries:

- UnrealCV
- Numpy
- Math
- Pillow
- StringIO
- os

User guide

First steps

The simulator is separated in two parts, one for central cameras and other for no central cameras, both inside the same folder. To work properly with this simulator, you have to copy the simulator folder called "Pack" in the same directory where the UnrealCV executable is located. In the example we will see how this path looks like.

Once we have the folder in the correct directory, we have to copy the configuration file of UnrealCV for the kind of cameras we want to make the images (Central or No-Central cameras). Inside each folder we have a different configuration file, "unrealcv.ini", which we should copy in the same directory as the executable of our scenario.

Central cameras

This cameras are modeled acording different mathematical models. These models are explained with detail in the papers of Jesus Bermudez and in the master thesis of Bruno Berenguel.

Once known the models, we start to build the images of this simulator mapping our environment in a sphere arround the camera. As an sphere is difficult to obtain from planar images, we aproximate it with a cube-map formed by 6 images in the main directions (the cartesian axis X, Y, Z). We build a cube-map for each location in our scenario where we want to make an image. To do so with our simulator, we first need to set the locations where we want to take the images. Editing the file 'camera-loc.txt', we set the different locations where we are going to get the cube-maps. In this file we have to write the coordinates (x, y, z) in one line and separated by white spaces for each location. Once set the different locations, we run the scenario of UnrealCV and the python script 'get-central-images.py', following the instructions, in the example we will explain the meaning of the instructions and it's purpose. This script will get three kind of images and cubemaps for each location, that will be used to build the rest of the images of this simulator, and store them in different folders, according to type and location. After the script has finished, we can close the scenario of UnrealCV.

We have seen that we have one script to build the cube-maps and interact with the plugin UnrealCV. Now we are going to build the omnidirectional images using the script 'simulator.py'. This script will call for other scripts which contain the mathematical models for the different cameras we are simulating. Through this simulator script we will have access to some parameters in the building of the images, such as resolution, locations we want to make, and main direction of the camera (set in the center of the image). If you want to change other parameters, you will have to enter into the scripts and change them manually.

When we run the simulator, a menu will appear, where we will be able to select which image we want to build. After we have choosen the image, we can choose between leave the default parameters of the script, then the program will start buildind the image, or change some of them according to your requirements. The program will ask us to enter the parameters sequentialy and

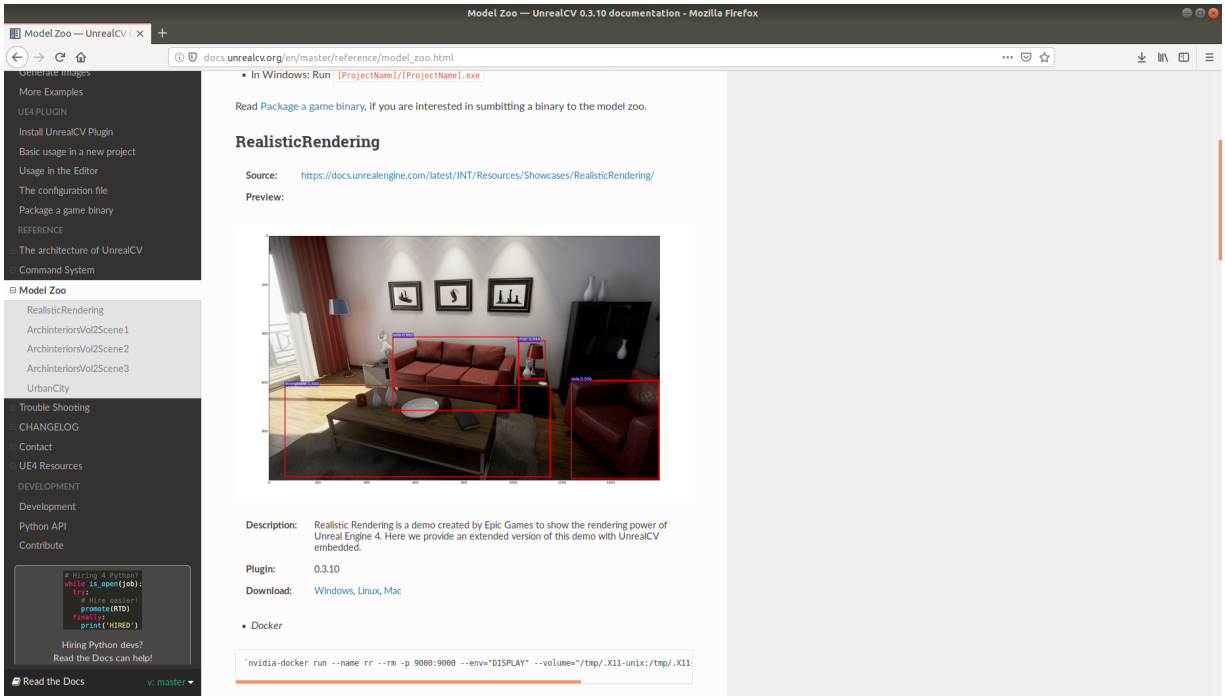
after all parameters are selected, it will start to build the image. In the example you will see how this menu looks like and how to navigate through it.

Run Test

Copy the configuration file where the scenario executable is located
Run the UnrealCV scenario and the python script 'get-central-images.py'
Once it finishes, close the UnrealCV scenario.
Run the python script 'simulator.py', select the scenario RR and 'test'
Compare the images with the model images

Step-by-Step example for central cameras

First thing we are going to do is download a scenario from UnrealCV In this example, it will be RealisticRendering.



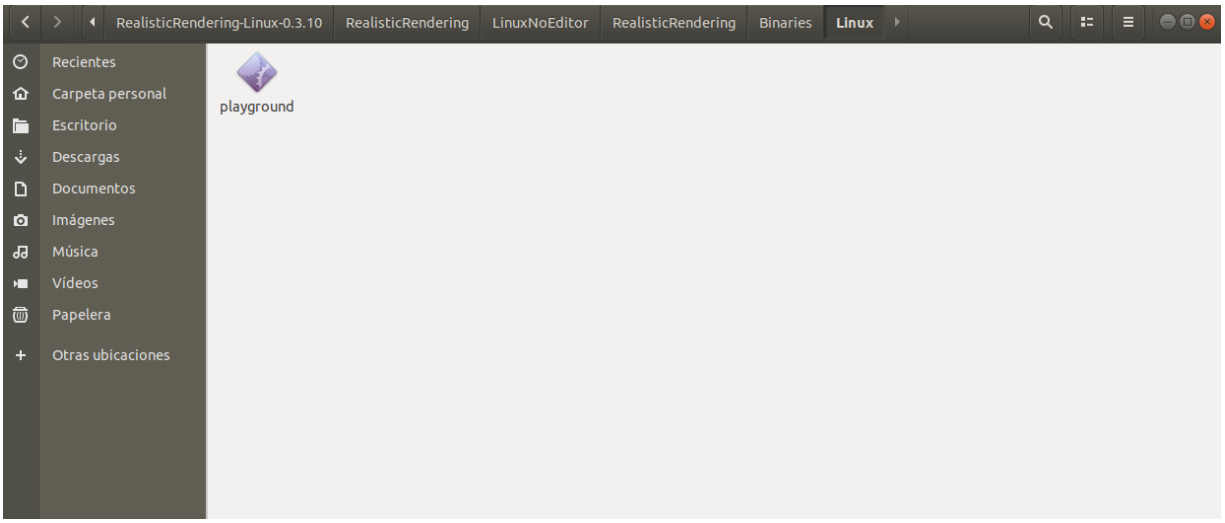
Once we have downloaded the scenario and unzipped it, we go to the directory of the executable and copy the folder of our simulator. Then, we copy the configuration file of UnrealCV.

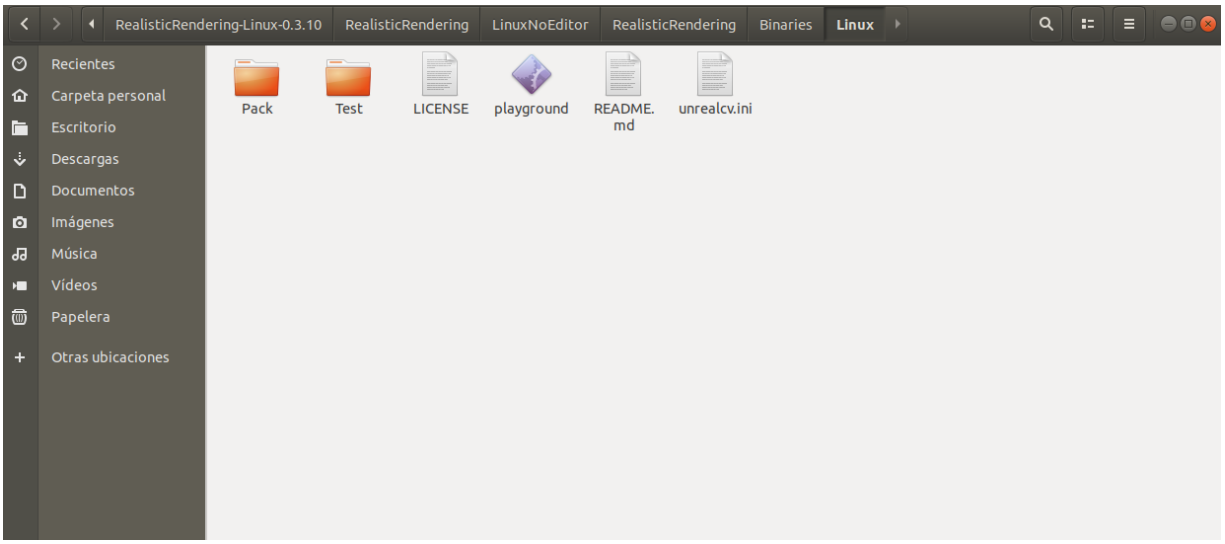
We write in çamera-loc.txt"the locations where we want to take the images:

$$\begin{matrix} 0 & 0 & 150 \\ 100 & 0 & 200 \\ 0 & 200 & 100 \end{matrix}$$

We have set 3 different locations inside our scenario.

Once done the set-up, we run the UnrealCV scenario and the python script 'get-central-images.py'. This proccess can take around 25 seconds for each location. When we run this





script, it will ask us the maximun depth and the starting index. The maximun depth will give a range where we can be able to see the depth images in a grey scale. If this maximun depth is too close, everything will be seen white, while if it is too far, everything will be seen black. We must adapt this value to our scene. On the other hand, the starting index is used if we already have a previous data set and we don't want to overwrite it.

PATH = REALISTICRENDERING-LINUX-0.3.10/REALISTICRENDERING/LINUXNOEDITOR/REALISTICRENDERING/BINARIES/LINUX
PATH$ ./PLAYGROUND
PATH/PACK/PACK-CENTRAL$ PYTHON GET-CENTRAL-IMAGES.PY
INTRODUCE MAXIMUN DEPTH IN METERS: 10.0
INTRODUCE STARTING INDEX: 1

Once it finishes, a new folder should have appeared called central, as we are going to use this images to create central camera's images. Inside this folder we have the images that we have made and the cube maps for each location and each type of images: lit: usual camera image

object-mask: each object has a different and plain color depth: give a .npy file with the distance to the camera and an aproximation image in grey-scale.

Now we can close the UnrealCV scenario.

First we are going to get hyper-catadioptric images. To do so, we must run the python script 'simulator.py', enter the scenario in which we want to build the images, RR in this case, and select the Catadioptric image.

```
$ PYTHON SIMULATOR.PY
(EQ)UIRECTANGULAR IMAGE
(CY)LINDRIC IMAGE
(FI)SH EYE IMAGE
(CA)TADIOPTRIC IMAGE
(SC)ARAMUZZA IMAGE
(S)CENARIO
(CFL)
(T)EST
(E)XIT
WHAT DO YOU WANT TO DO?: CA
```

Once we do so, it will ask us if we want to use the default set-up or enter new parameters. If we use the default set-up, it will use the parameters which are default in the script, which we can change manually. If we want to use other parameters or have control of them, we should say "No" and enter them sequentially as the simulator ask for them.

```
USE FILE SET-UP (YES) OR ENTER NEW PARAMETERS(NO): NO
ENTER WIDTH: 1024
ENTER HEIGHT: 1024
ENTER MODE LIT/OBJECT-MASK/DEPTH: LIT
ENTER FIRST LOCATION: 1
ENTER NUMBER OF LOCATIONS: 2
ENTER SYSTEM PARA/HYPER/PLANAR: HYPER
ENTER DISTANCE BETWEEN CAMERA AN MIRROR: 0.75
ENTER LATUS RECTUM: 0.1
ENTER VIEW DIRECTION AXIS: X
ENTER AXIS SIGN POS/NEG: POS
DATA SET? TRUE/FALSE: TRUE
```

In this example we have set the resolution of the image in 1024x1024 pixels, we will get the first and second location in the "lit" mode (usual camera images) for a hyper-catadioptric camera system where the mirror is located 0.75 meters from the camera and its latus rectum is 0.1 meters. The camera is looking into the "x" axis, to the positive direction, which means as we have a mirror that we will see into the opposite direction. Setting the Data set as "False", the simulator will show us the images once they are build and stored. If this parameter is set as "True", the simulator will store the image without showing them to the user, which is very convenient if we want to make lots of images.

In this point, we have to realice that new folders have appeared in our working directory with the names of the images we have build. The images we have build will be stored inside these folders and named according to the mode and location.

Step-by-Step example for no central cameras

Coming next