3DScanner

Intel RealSense L515-LiDAR

Sadržaj

1.0 Verzije i biblioteke

- *Python 3.7.9*, poslednja kompatibilna verzija sa dole navedenim i trenutno dostupnim *PyPI* bibliotekama.
- *Realsense2,* omogućava nam integraciju sa *Intel RealSense SDK 2.0* paketom.
- *Open3D-Python 0.7.0,* biblioteka koja nam omogućava pristup raznim strukturama/modelima podataka (*PointCloud*, *STL*...) kao i funkcijama vizualizacije istih.
- *OpenCV-Python 4.5.3.56, Open Source Computer Vision Library,* biblioteka koja nam omogućava pristup velikom broju *Vison* algoritama i filtera.

2.0 Intel Realsense L515 LiDAR kamera

- L515 je kamera bazirana na LiDAR (*Light Detection and Ranging*) tehnologiji za razliku od svojih *STEREO-VISION* prethodnika. Obe tehnologije se koriste za detekciju objekata. *LiDAR* kamere koriste lasere pomoću kojih na osnovu brzine prijema emitovanog lasera **određuju** udaljenost objekta (princip sličan sonarima). *LiDAR* kamere karakteriše visoka preciznost (merenja u cm, do 9m) dok im je najveca mana to što zavise od same reflektivnosti objekata kao i vremena, na primer ukoliko je kišno može doći do refleksije koja nastupa pre samog dodira lasera i posmatranog objekta čime dobijamo potpuno netačna očitavanja.
- *STEREO-VISION* (*D400+*) kamere **predpostavljaju** udaljenost do nekog objekta pomoću barem dve fotografije iz različitih uglova. Ne pružaju tačne udaljenosti nego estimacije.

3.0 Intel RealSense SDK 2.0

- Automatizovano podešavanje odgovarajuće rezolucije u zavisnost od modela kamere

```
# Get device product line for setting a supporting resolution
pipeline_wrapper = rs.pipeline_wrapper(pipeline)
pipeline_profile = config.resolve(pipeline_wrapper)
device = pipeline_profile.get_device()
device_product_line = str(device.get_info(rs.camera_info.product_line))
```

```
if device_product_line == 'L500':
    config.enable_stream(rs.stream.color, 960, 540, rs.format.bgr8, 30)
else:
    config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
```

- Lako razdvajanje *Color* i *Depth Frame*-ova

```
# Wait for a coherent pair of frames: depth and color
        frames = pipeline.wait_for_frames()
        depth_frame = frames.get_depth_frame()
        color_frame = frames.get_color_frame()
```

3.1 Open Computer Vision Library

- Omogućava jednostavno očitavanje slike sa kamere/fajla, obradu i prikaz

```
while True:
    ret, frame = cap.read()
    width = int(cap.get(3))
    height = int(cap.get(4))
    canvas = np.zeros(frame.shape, np.uint8)
    cv2.imshow('openCVCameraDEMO', canvas)
    if cv2.waitKey(1) == ord('q'):
        break
```

- Jednostavno oslobađanje zauzete kamere

```
cap.release()
cv2.destroyAllWindows()
```

- Jednostavna primena *Kernel* matrica radi filtriranja

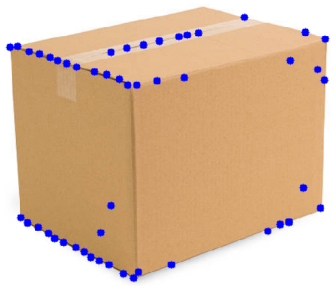
```
img = cv2.imread("Resursi/deer.jpg", 1)
kernel = np.ones((10, 10), 'uint8')
DilationFilter = cv2.dilate(img, kernel)
cv2.imshow('DilationDemo', DilationFilter)
cv2.waitKey(0)
cv2.destroyAllWindows()

kernelErode = np.ones((5, 5), np.uint8)
ErodeFilter = cv2.erode(img, kernelErode)
cv2.imshow("ErodeDemo", ErodeFilter)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Jednostavna primena prethodno treniranih kaskadnih fajlova za detekciju i ekstrakciju

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
img_grayscale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(img_grayscale, 1.3, 5)
```

- Jednostavna primena Thresholding-a radi lakšeg procesiranja

```
corners = cv2.goodFeaturesToTrack(img_grayscale, 100, 0.01, 10)
corners = np.int0(corners)
```



3.2 Open3D

- Jednostavno generisanje RGB+D fotografija

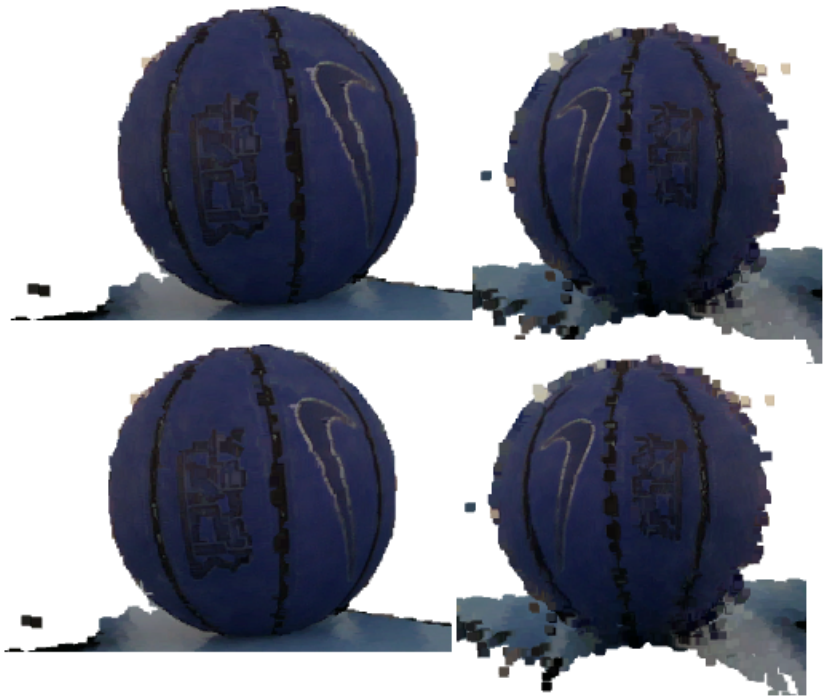
```
self.depth_frame_open3d = o3d.geometry.Image(self.depth_image)
self.color_frame_open3d = o3d.geometry.Image(self.color_image)

self.rgbd_image = create_rgbd_image_from_color_and_depth(self.color_frame_open3d,
                                                         self.depth_frame_open3d,
                                                    convert_rgb_to_intensity=False)
```

- Jednostavno generisanje PointCloud-a na osnovu RGB+D fotografija

```
main_pcd = o3d.geometry.PointCloud() #Empty
main_pcd = o3d.geometry.create_point_cloud_from_rgbd_image(self.rgbd_image, self.intrinsic)
```

- Jednostavno učitavanje i prikaz PointCloud-a

```
pcd = read_point_cloud("my_cloud.ply", format='xyz')
o3d.visualization.draw_geometries([pcd])
```



3.3 Rotacione/Transformacione matrice

- Matrice translacije, transliraju model po zadatoj osi

```
def Translation(x, y, z):
    return [[1, 0, 0, 0],
            [0, 1, 0, 0],
            [0, 0, 1, 0],
            [x, y, z, 1]]
```

- Matrice skaliranja, skaliraju model u zavisnosti od zadate ose

```
def Scale(x, y, z):
    return [[x, 0, 0, 0],
            [0, y, 0, 0],
            [0, 0, z, 0],
            [0, 0, 0, 1]]
```

- Matrice rotacije, rotiraju model na osnovu zadatog ugla

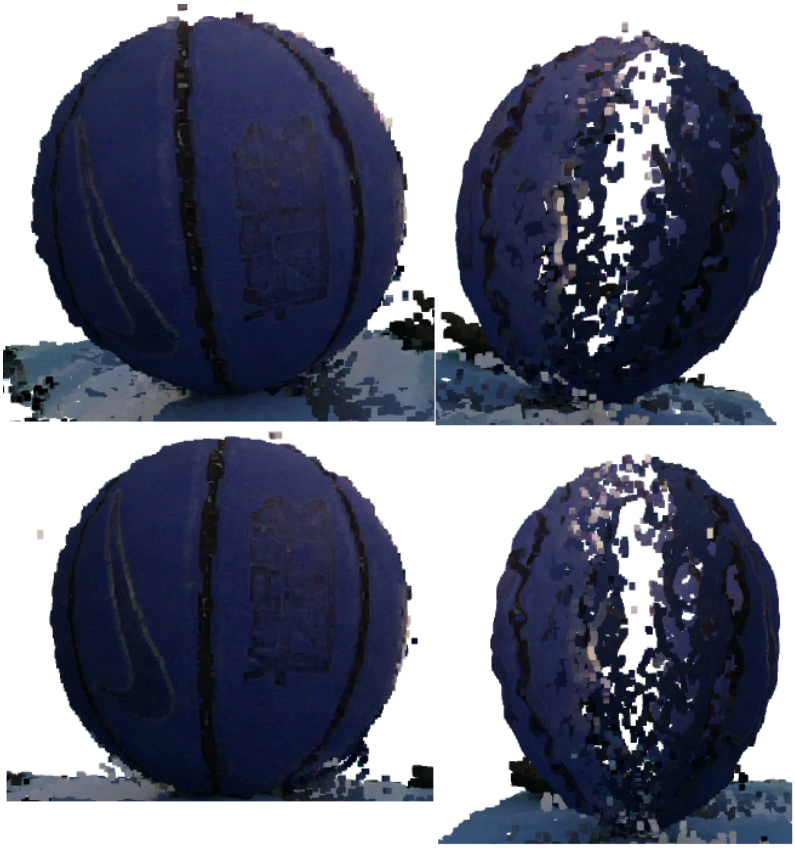
```
def X_Rotation(angle):
    return [[1, 0, 0, 0],
            [0, np.cos(angle), np.sin(angle), 0],
            [0, -np.sin(angle), np.cos(angle), 0],
            [0, 0, 0, 1]]
```

```
def Y_Rotation(angle):
    return [[np.cos(angle), 0, -np.sin(angle), 0],
```

```
            [0, 1, 0, 0],
            [np.sin(angle), 0, np.cos(angle), 0],
            [0, 0, 0, 1]]
```

```
def Z_Rotation(angle):
    return [[np.cos(angle), np.sin(angle), 0, 0],
            [-np.sin(angle), np.cos(angle), 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 1]]
```

- Primenom matrica transformacija možemo postići preklapanja više modela kao i njihovo spajanje radi dobijanja potpunog, “spremnog za štampu” modela.



3.4 Post-Processing filteri

Prilikom procesiranja fotografija objekata mogu se koristiti dole navedeni filteri.

- Disparity transform - omogućava transformaciju između depth framova i različitih domena i aplikabilna je samo za stereo zasnovane depth domene.
- Decimation filter - redukuje kompleksnost depth slike. Korišćenjem ovog filtera dolazi do proprcionalnog skaliranja obe dimenzije u cilju dostizanja adekvatnog aspect ratio-a

(odgovarajuće širine i visine slike). Ovaj filter ima u cilju pojednostavljenje kompleksnoti radi lakseg procesiranja slike.

- Temporal filter - koristi se prilikom prikazivanja postojanosti depth podataka uz pomoć manipulacije nad pixel vrednostima zasnovanim na osnovu prethodnog frejma. Filter se istovremeno primenjuje samo nad jednim podatkom prilagođavajući vrednosti i istovremeno čuvajući vrednosti u istoriji izmena. U slučajevima kada nedostaju podaci ili ukoliko je filter primenjen nevalidno koristi se korisničko-definisani procena postojanosti na osnovu koje se odlučuje da li će se oštećeni podatak ispravljati i uklapati sa već postojećim podacima.
- Holes filing filter - implementira nekoliko metoda uz pomoć kojih nadopunjuje delove fotografija koji nedostaju. Filter uzima susedne, “komšijske” pixele (gornji, donji, desni, levi) i selektuje jedan od njih na osnovu korisnički-definisanih pravila. Metode koje ovaj filter implementira su sledeće :
    - ***fill_from_left*** - koristi vrednost levog ”komšijskog” pixela da bi dopunio postojeću prazninu.
    - ***farest_from_around* -** koristi vrednost ”komšijskog” pixela koji je najudaljeniji od senzora.
    - ***nearest_from_around* -** koristi vrednost ”komšijskog” pixela koji je najbliži senzoru.

3.5 Poboljšanja

- Ukoliko uvedemo konstantnu vrednost rastojanja između kamere i modela, povećamo broj fotografija i smanjimo ugao rotiranja možemo dobiti mnogo kvalitetnije/preciznije modele
- Mogućnost uvođenja rotacione *Arduino* platforme sa malim step motorićem radi dobijanja konstante udaljenosti i precizno definisanog ugla rotacije motora/modela.
- Primena ICP algoritma za spajanje modela