Misure e strumentazioni per l'automazione

Pepper Robot Door Detection

Implementazione nodo ROS per la detection di porte e maniglie

Prof. Alessandro Freddi
Dott. Ing. Daniele Proietti Pagnotta

Daniele Delli Rocili
Fabio Morganti
Francesco Giostra

Indice

1 Introduzione

Uno dei principali problemi che hanno i robot è l'impossibilità di accedere a stanze diverse attraverso l'identificazione e l'apertura di porte. Dato il seguente problema, nasce la costruzione del nostro nodo ROS, volto ad identificare porte e maniglie tramite l'acquisizione di immagini dalla telecamera frontale del Pepper robot, le quali saranno processate da una rete neurale per l'identificazione delle coordinate spaziali di porte e maniglie.

2 Strumentazione

2.1 Pepper robot

Pepper è un robot umanoide prodotto da Softbank Robotics, il quale presenta le seguenti caratteristiche tecniche:

2.1.1 Dimensioni

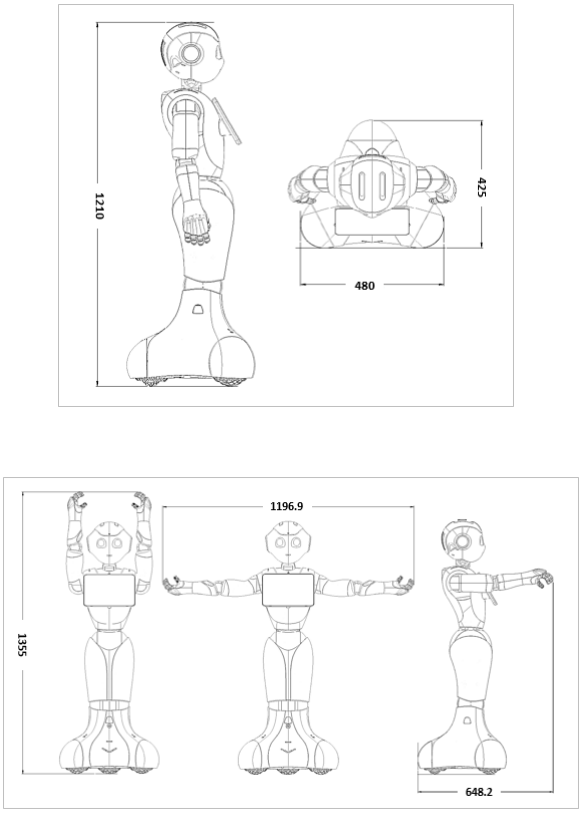

Figura 1. Dimensioni robot

Il Pepper presenta un'altezza minima di 1210 mm fino a raggiungere una massima di 1355 mm, una larghezza minima di 480 mm e una massima di 1196.9 mm, una profondità minima di 425 mm e una massima di 648.2 mm.

2.1.2 Batteria

Il robot è alimentato attraverso una batteria a ioni di litio, con un voltaggio nominale di 26.46V e un amperaggio minimo di 29.5Ah.

2.1.3 Interazioni e sensori

L'interazione con l'esterno avviene mediante l'uso di telecamere e sensori, in particolare:

- 2D Cameras
- Inertial unit
- Lasers
- Infra-Red
- Sonars
- MRE (Magnetic Rotary Encoders)

2.1.4 Motori

Per il movimento all'interno dell'ambiente di lavoro vengono usati tre motori Brushless DC, modello EC45_70W.

2.2 ROS

Il Robot Operating System (ROS) è un framework flessibile per la scrittura di software per robot. Fornisce le stesse funzioni offerte da un sistema operativo su un cluster composto da diversi tipi di elaboratori. ROS fornisce i servizi standard di un sistema operativo, come: astrazione dell'hardware, controllo dei dispositivi tramite driver, comunicazione tra processi, gestione delle applicazioni (package) e altre funzioni di uso comune. Un insieme di processi all'interno di ROS si possono rappresentare in un grafo come dei nodi che possono ricevere, inviare e multiplexare i messaggi provenienti da e verso altri nodi, sensori e attuatori.

Figura 2. Logo ROS

Nell'architettura di ROS i software possono essere raggruppati in tre categorie:

- strumenti per lo sviluppo e pubblicazione di software basato su ROS;
- librerie utilizzabili dai processi ROS client come roscpp, rospy e roslisp;
- pacchetti (packages) contenenti applicazioni e codice che usa una o più librerie per processi ROS client.

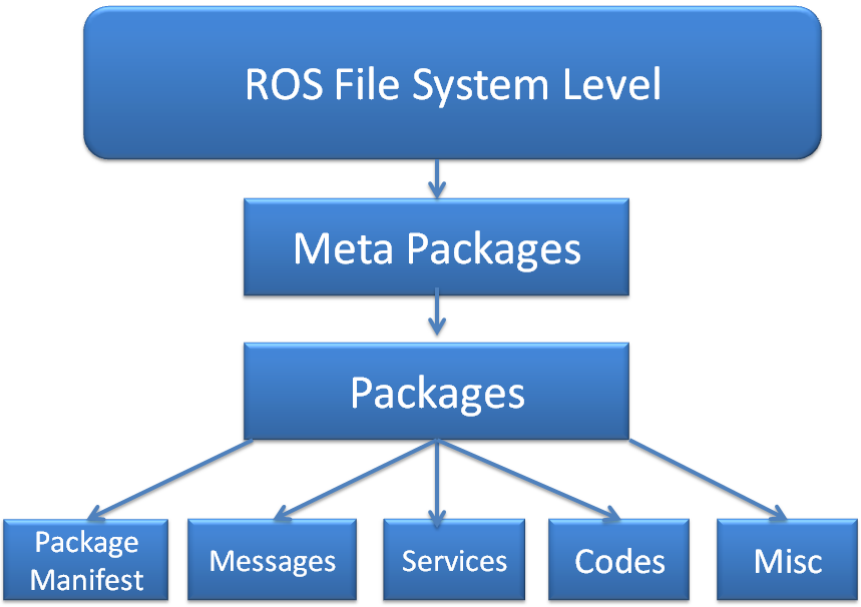

Figura 3. Ros File System

La versione utilizzata in questo progetto è ROS Kinetic.

2.3 Gazebo

Figura 4. Logo Gazebo

Gazebo è un simulatore di robotica 3D open source, con la capacità di simulare in modo accurato ed efficiente popolazioni di robot in ambienti interni ed esterni complessi; puoi valutare e testare il tuo robot in scenari difficili o pericolosi senza alcun danno al tuo robot. È simile ai motori di gioco , ma produce simulazioni migliori e offre una suite di sensori e interfacce sia per gli utenti che per i programmi.
Comprende i seguenti componenti principali:

- World files: contengono tutti gli elementi di una simulazione, inclusi robot, luci, sensori e oggetti statici.
- Models: rappresentano i singoli elementi.
- gzserver: legge il file world per generare e popolare un mondo.
- gzclient: si connette a gzserver e visualizza gli elementi.
- gzweb: versione web di gzclient, utilizzando WebGL.

La versione utilizzata nel nostro progetto è Gazebo 7.16.1.

2.4 Google Colab

Figura 5. Logo Colab

Colaboratory o, in breve, Colab è un prodotto di Google Research. Colab permette a chiunque di scrivere ed eseguire codice Python arbitrario tramite il browser ed è particolarmente adatto per machine learning, analisi dei dati e formazione. Più tecnicamente, Colab è un servizio di blocchi note Jupyter, il cui utilizzo non richiede alcuna configurazione, che fornisce l'accesso gratuito alle risorse di calcolo, comprese le GPU.

3 Dataset

Per l'addestramento della rete neurale, volta all'individuazione di porte e maniglie è stato utilizzato un dataset di 1213 immagini, reperito dal repository Github: https://github.com/MiguelARD/DoorDetect-Dataset.
Il dataset prevede per ogni immagine un file di testo contenente le informazioni relative ai bounding box: classe, coordinate X e Y centrali, larghezze e altezza. Oltre alle classi relative a porte e maniglie, il dataset presenta altre due classi, che indicano la presenza di armadietti e frigoriferi.
Di seguito un esempio di un'immagine di una porta con il suo relativo label.

Figura 6. porta

1	0.661328	0.880556	0.060156	0.022222
0	0.508984	0.697917	0.399219	0.604167

Tale dataset è stato successivamente ampliato da ulteriori 85 foto di porte del dipartimento, in quanto sarà l'ambiente principale di lavoro del Pepper. Dato che queste ultime immagini erano sprovviste di label, abbiamo dovuto annotarle attraverso l'utilizzo del sito https://www.makesense.ai/.

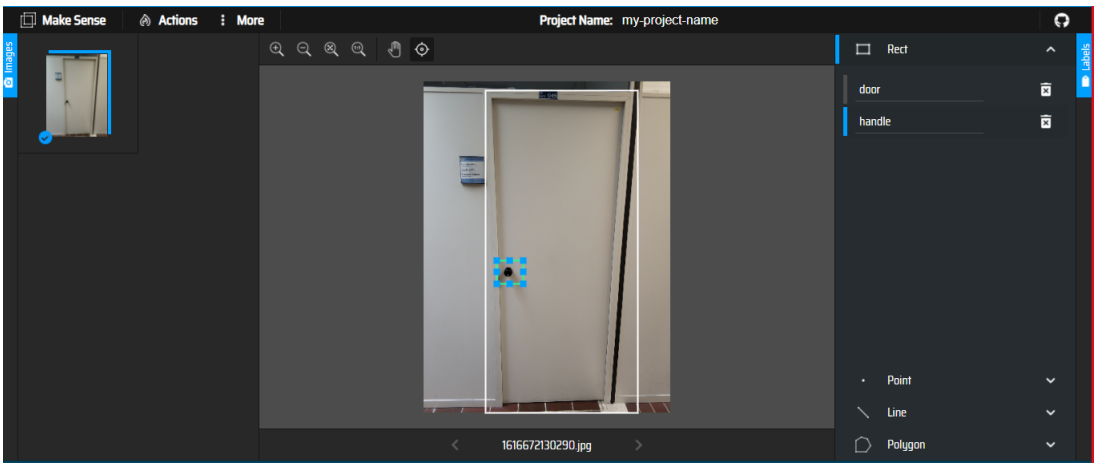

Figura 7. makesense

Dopo aver creato i label, Makesense genera un file di testo per ogni immagine, con lo stesso formato del dataset precedente.

3.1 Creazione dataset

Per l'allenamento della nostra rete era necessario avere un file di testo con la seguente struttura: path_immagine, Xmin, Ymin, Xmax, Ymax, classe. Per la crazione del dataset abbiamo sfruttato le funzionalità di python per convertire tutti i file relativi ai label delle immagini, in un unico file con la struttura precedentemente descritta.
Il primo passo per ottenere il nostro dataset, precedentemente descritto, è stato quello di trasformare i label del dataset di github.

```
import os
from PIL import Image
import cv2
```

```
path_labels = '/content/drive/MyDrive/Datasets/door_dataset/labels'
path_new_labels = '/content/drive/MyDrive/Datasets/door_dataset/labels2'
path_images = '/content/drive/MyDrive/Datasets/door_dataset/images'
```

```
for path in os.listdir(path_labels):
  f = open(path_labels+'/'+path, "r")
  f2 = open(path_new_labels+'/'+path, "w")
  for line in f:
    line_split = line.split(' ')
    class_number = line_split[0]
    x_center = float(line_split[1])
    y_center = float(line_split[2])
    width = float(line_split[3])
    height = float(line_split[4])
    x_min = x_center - (width/2)
    y_min = y_center - (height/2)
    x_max = x_center + (width/2)
    y_max = y_center + (height/2)
    x_min = "{:.6f}".format(x_min)
    y_min = "{:.6f}".format(y_min)
    x_max = "{:.6f}".format(x_max)
    y_max = "{:.6f}".format(y_max)
    f2.write(class_number+' '+str(x_min)+' '+str(y_min)+' '+str(x_max)+'
        '+str(y_max)+'\n')
  f2.close()
  f.close()
```

Dal momento che i labels di ogni immagine erano contenuti in file diversi, per nostre esigenze è stato necessario raggruppare tutte le informazioni in un unico file.

```
path_images = '/content/drive/MyDrive/Datasets/door_dataset/images'
path_new_labels = '/content/drive/MyDrive/Datasets/door_dataset/labels2'
path_dataset = '/content/drive/MyDrive/Datasets/dataset'
```

```
dataset = open(path_dataset+'/'+'Test_porte.txt', "w")
for image in os.listdir(path_images):
  image_name = image
  im = Image.open(path_images+'/'+image_name)
  dw, dh = im.size
  image = image.split('.')
  f = open(path_new_labels+'/'+image[0]+'.txt', "r")
  strings = []
  for line in f:
    line_split = line.split(' ')
    class_number = line_split[0]
    x_center = line_split[1]
    y_center = line_split[2]
    width = line_split[3]
```

```
    height = line_split[4]
    if len(height) > 8:
      height = height[:len(height)-1]
    x_min = float(x_center)-(float(width)/2)
    y_min = float(y_center)-(float(height)/2)
    x_max = float(x_center)+(float(width)/2)
    y_max = float(y_center)+(float(height)/2)
    x_min = int(x_min*dw)
    y_min = int(y_min*dh)
    x_max = int(x_max*dw)
    y_max = int(y_max*dh)
    if (x_max > dw):
      x_max = dw
    if (y_max > dh):
      y_max = dh
    strings.append('
        '+str(x_min)+','+str(y_min)+','+str(x_max)+','+str(y_max)+','+str(class_number))
  f.close()
  strings2 = ''.join(strings)
  line_data = str(path_images+'/'+str(image_name)) + strings2 + str('\n')
  dataset.write(line_data)

dataset.close()
```

Successivamente abbiamo integrato al file unico, precedentemente descritto, i labels relativi alle porte del dipartimento, modificandone il formato.

```
path_images = '/content/drive/MyDrive/Datasets/door_dataset_freddi/images'
path_new_labels = '/content/drive/MyDrive/Datasets/door_dataset_freddi/labels'
path_dataset = '/content/drive/MyDrive/Datasets/dataset'
```

```
dataset = open(path_dataset+'/'+'dataset2.txt', "a")
for image in os.listdir(path_images):
  image_name = image
  im = Image.open(path_images+'/'+image_name)
  dw, dh = im.size
  image = image.split('.')
  f = open(path_new_labels+'/'+image[0]+'.txt', "r")
  strings = []
  for line in f:
    line_split = line.split(' ')
    class_number = line_split[0]
    x_center = line_split[1]
    y_center = line_split[2]
    width = line_split[3]
    height = line_split[4]
    if len(height) > 8:
      height = height[:len(height)-1]
    x_min = float(x_center)-(float(width)/2)
    y_min = float(y_center)-(float(height)/2)
    x_max = float(x_center)+(float(width)/2)
    y_max = float(y_center)+(float(height)/2)
    x_min = int(x_min*dw)
    y_min = int(y_min*dh)
    x_max = int(x_max*dw)
    y_max = int(y_max*dh)
    if (x_max > dw):
      x_max = dw
    if (y_max > dh):
      y_max = dh
```

```
    strings.append('
        '+str(x_min)+','+str(y_min)+','+str(x_max)+','+str(y_max)+','+str(class_number))
  f.close()
  strings2 = ''.join(strings)
  line_data = str(path_images+'/'+str(image_name)) + strings2 + str('\n')
  dataset.write(line_data)

dataset.close()
```

Il dataset è stato successivamente diviso in Train e Test, rispettivamente 80% e 20%.

```
import pandas as pd
from sklearn.model_selection import train_test_split
data = pd.read_csv('/content/drive/MyDrive/Datasets/dataset/dataset2.txt', sep="\n",
    header=None)
```

```
train, test = train_test_split(data, test_size=0.2, random_state=78)
```

```
train.to_csv("/content/drive/MyDrive/Datasets/dataset/train_nuovo.txt", header=None,
    index=None)
test.to_csv("/content/drive/MyDrive/Datasets/dataset/test_nuovo.txt", header=None,
    index=None)
```

Per un miglior riscontro sui risultati, abbiamo deciso di creare un dataset di test con solo porte e maniglie. Di seguito un esempio di immagine.

Figura 8. porta test

```
path_images = '/content/drive/MyDrive/A.Misure progetto/New Dataset/images'
path_new_labels = '/content/drive/MyDrive/A.Misure progetto/New Dataset/labels'
path_dataset = '/content/drive/MyDrive/Datasets/dataset'
```

```
dataset = open(path_dataset+'/'+'Test_porte.txt', "w")
for image in os.listdir(path_images):
  image_name = image
  im = Image.open(path_images+'/'+image_name)
  dw, dh = im.size
  image = image.split('.')
  f = open(path_new_labels+'/'+image[0]+'.txt', "r")
  strings = []
  for line in f:
    line_split = line.split(' ')
    class_number = line_split[0]
    x_center = line_split[1]
    y_center = line_split[2]
    width = line_split[3]
    height = line_split[4]
    if len(height) > 8:
      height = height[:len(height)-1]
```

```
    x_min = float(x_center)-(float(width)/2)
    y_min = float(y_center)-(float(height)/2)
    x_max = float(x_center)+(float(width)/2)
    y_max = float(y_center)+(float(height)/2)
    x_min = int(x_min*dw)
    y_min = int(y_min*dh)
    x_max = int(x_max*dw)
    y_max = int(y_max*dh)
    if (x_max > dw):
      x_max = dw
    if (y_max > dh):
      y_max = dh
    strings.append('
        '+str(x_min)+','+str(y_min)+','+str(x_max)+','+str(y_max)+','+str(class_number))
  f.close()
  strings2 = ''.join(strings)
  line_data = str(path_images+'/'+str(image_name)) + strings2 + str('\n')
  dataset.write(line_data)

dataset.close()
```

4 YOLOv3

Yolo è stato sviluppato da Redmon e Farhadi nel 2015, durante il loro dottorato. Il concetto è di ridimensionare limmagine in modo da ricavarne una griglia di quadrati. Nella v3, YOLO fa predizioni su 3 diverse scale, riducendo limmagine rispettivamente di 32, 16 e 8, allo scopo di rimanere accurata anche su scale più piccole (le versioni precedenti avevano dei problemi con le immagini piccole). Per ciascuna delle 3 scale, ogni cella è responsabile della predizione di 3 bounding box, utilizzando 3 anchor. YOLOv3 differentemente dalle sue versioni precedenti, risulta essere migliore nel rilevamento di oggetti più piccoli. YOLOv3 utilizza una rete convoluzionale chiamata efficientNet, l'elemento costitutivo principale di questa rete è MBConv a cui viene aggiunta l'ottimizzazione di compressione ed eccitazione. MBConv è simile ai blocchi residui invertiti utilizzati in MobileNet v2. Questa struttura aiuta a ridurre il numero complessivo di operazioni richieste e le dimensioni del modello.

Figura 9. Architettura EfficientNet

4.1 Analisi comparativa

YOLO v3 si comporta alla pari con altri rilevatori all'avanguardia come RetinaNet, pur essendo notevolmente più veloce, al benchmark COCO mAP 50. È anche migliore di SSD e le sue varianti. Di seguito un confronto delle prestazioni.

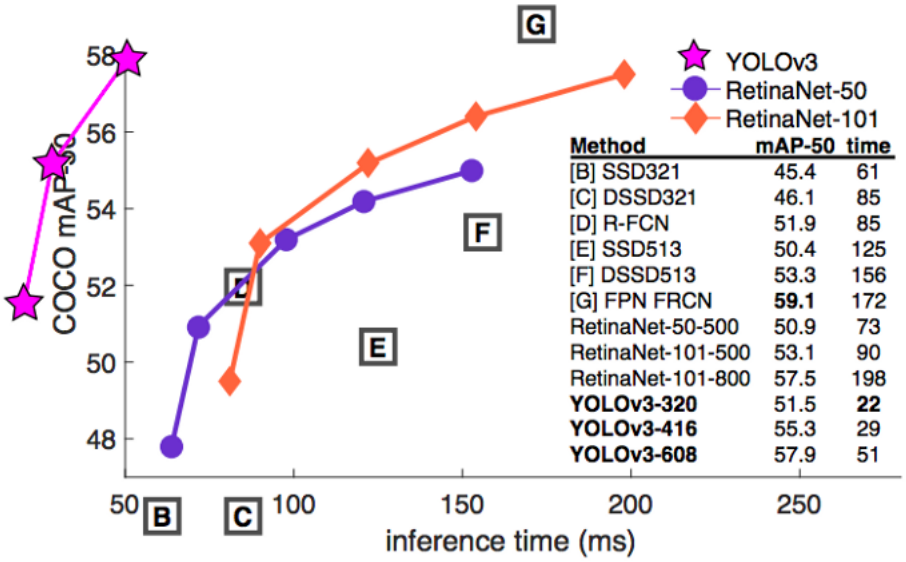

Figura 10. Comparativa YOLO

4.2 Training

Per l'addestramento della rete è stato utilizzato il seguente repository GitHub: https://github.com/david8862/keras-YOLOv3-model-set.
Sviluppando il seguente codice.

```
! pip install Cython
! pip install -r requirements.txt
! pip install tensorflow==2.2.0
! pip install numpy==1.20.2
```

```
! python train.py --model_type=yolo3_efficientnet --transfer_epoch=20
    --total_epoch=200 --anchors_path=configs/yolo3_anchors.txt
    --annotation_file=/content/drive/MyDrive/Datasets/dataset/train_nuovo.txt
    --classes_path=class.txt --batch_size=8 --optimizer=rmsprop --decay_type=cosine
```

```
! python yolo.py --model_type=yolo3_efficientnet
    --weights_path=logs/000/ep058-loss29.434-val_loss29.567.h5
    --anchors_path=configs/yolo3_anchors.txt --classes_path=class.txt
    --model_image_size=416x416 --dump_model --output_model_file=model.h5
```

Per un corretto funzionamento della rete, la versione necessaria per Tensorflow è la 2.2.0 e per Numpy è la 1.20.2. Dopo numerosi test sperimentali, è stato deciso di utilizzare la seguente configurazione:

- model_type = yolo3_efficientnet
- transfer_epoch = 20
- total_epoch = 200
- batch_size = 8
- optimizer = rmsprop
- decay_type = cosine

L'addestramento della rete è stato interrotto manualmente all'epoca 68, ciò dovuto alla validation loss che dopo diverse epoche risultava pressoché costante.

4.3 Metriche

Le metriche di valutazione del test sono:

4.3.1 Precision

La Precision tenta di rispondere alla domanda: quale percentuale di identificazioni positive è effettivamente corretta?
La formula utilizzata per il suo calcolo è la seguente:

$$Precision = \frac{TP}{TP + FP}$$

- TP: True Positive
- FP: False Positive

4.3.2 Recall

La Recall tenta di rispondere alla domanda: quale percentuale di effettivi positivi è stata identificata correttamente?
La formula utilizzata per il suo calcolo è la seguente:

$$Recall = \frac{TP}{TP + FN}$$

- TP: True Positive
- FN: False Negative

4.3.3 mAP

Prima di poter calcolare l'mAP (mean Average Precision), necessitiamo dei valori della precision e della recall, in modo da poter calcolare il nostro AP per ogni classe.
Costruiamo un piano cartesiano, al quale associamo all'asse delle ascisse la recall e alle ordinate la precision; ottenendo così una curva, la quale area sottesa rappresenta l'Average Precision, il tutto ricalcolato per ogni classe. Infine calcoliamo la media algebrica di tutti i valori di AP ottenuti.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

5 Testing

La fase di testing consiste nel valutare le performance della rete con immagini mai analizzate in precedenza, tali immagini possiedo anch'esse dei Ground Truth che identificano le classi al loro interno, in modo da poter confrontare i valori ottenuti dalla predizione della rete con i valori reali. Il codice utilizzato per questa fase è il seguente:

```
! python eval.py --model_path=model.h5 --anchors_path=configs/yolo3_anchors.txt
    --classes_path=class.txt --model_image_size=416x416 --eval_type=VOC
    --iou_threshold=0.5 --conf_threshold=0.3
    --annotation_file=/content/drive/MyDrive/Datasets/dataset/Test_porte.txt
    --save_result
```

5.1 Testing completo

Nel testing completo è stato utilizzato il 20% del dataset iniziale. Ottenendo così i seguenti risultati:

Figura 11. Foto test cabinet

Figura 12. Foto test frigorifero

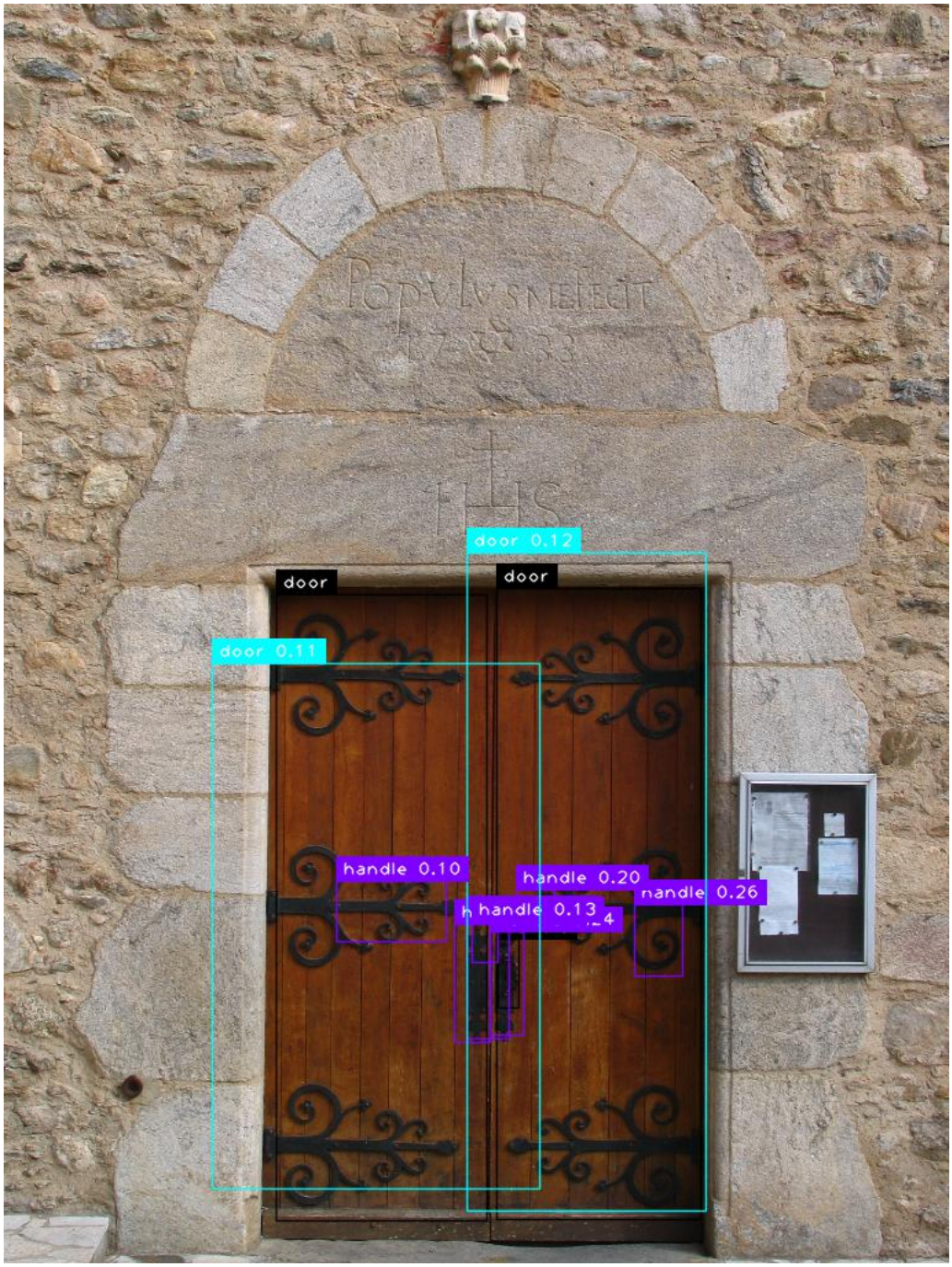

Figura 13. Foto test porta

Da questa immagine possiamo notare l'individuazione corretta delle porte, con un'eccessiva individuazione delle maniglie, dovuta ai decori della porta, ma con una soglia di confidenza molto bassa ($\simeq$0.1).

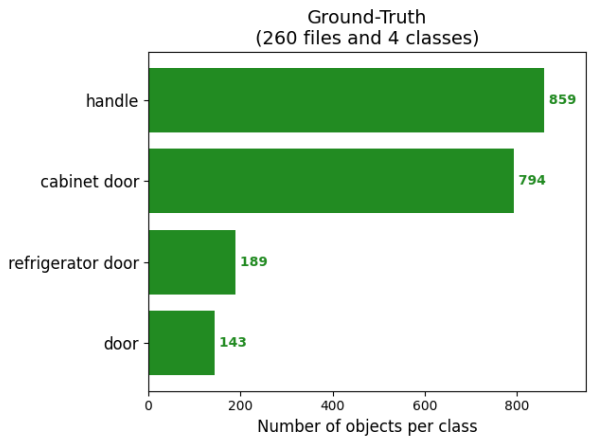

Figura 14. Ground Truth

Figura 15. Predicted objects

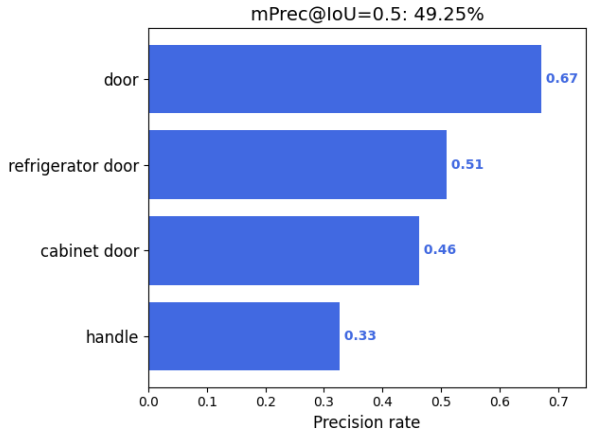

Figura 16. Precision

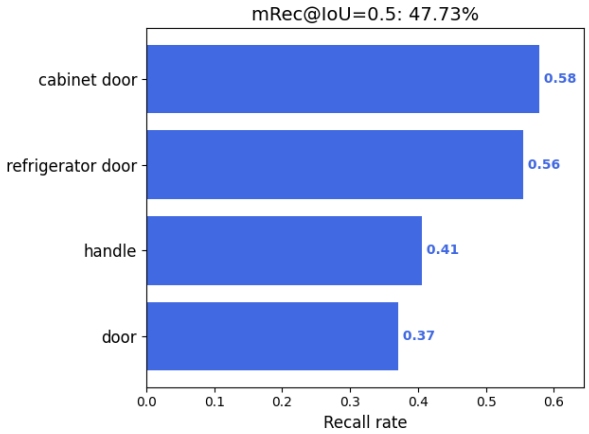

Figura 17. Recall

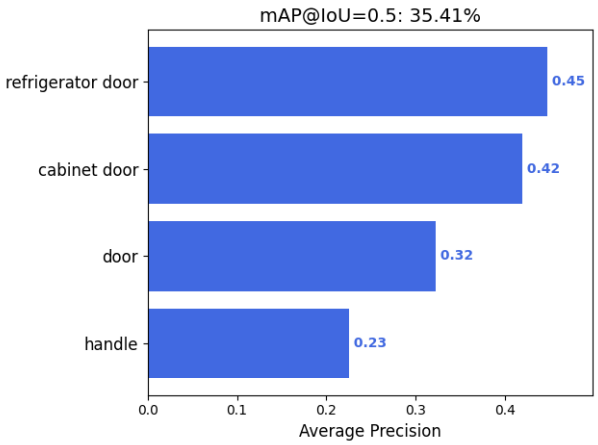

Figura 18. mean Average Precision

In questa fase il tempo d'inferenza per ogni immagine è stato di 0.42s.

5.2 Testing parziale

Il testing parziale, così chiamato per essere un testing eseguito solo su immagini contenenti porte e maniglie. Sono state utilizzate 40 immagini, di cui 30 reperite da internet e le rimanenti sono state riutilizzate le immagini del dipartimento, utilizzate nel testing precedente.

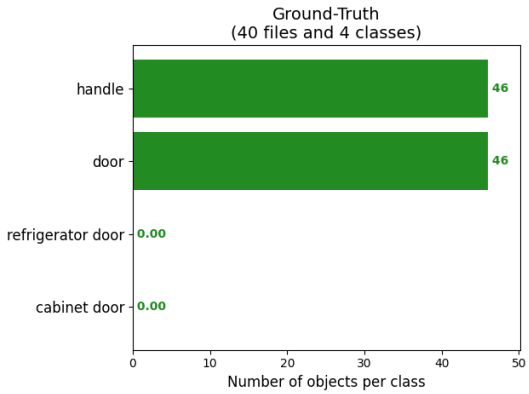

Figura 19. Ground Truth

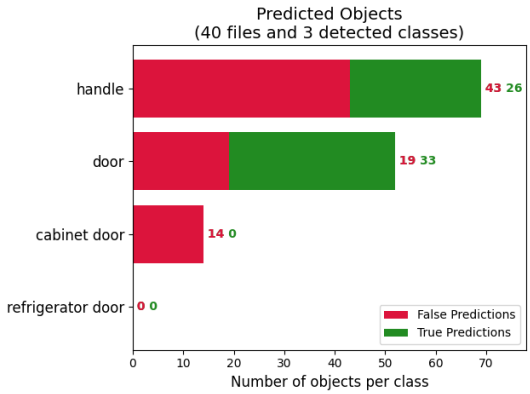

Figura 20. Predicted objects

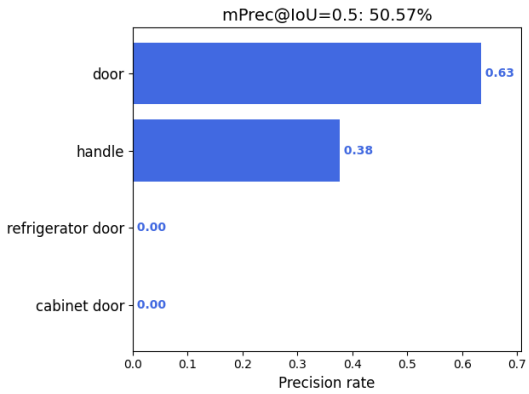

Figura 21. Precision

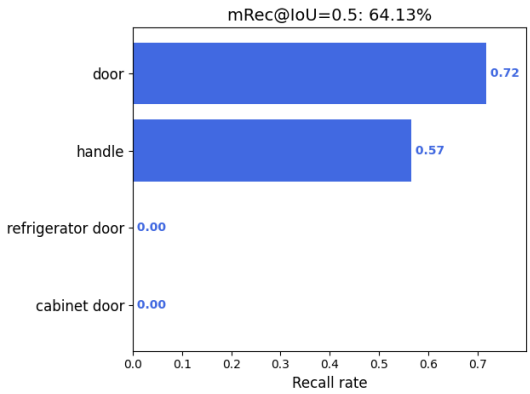

Figura 22. Recall

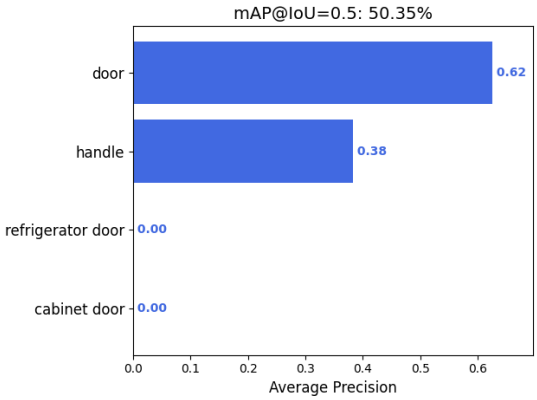

Figura 23. mean Average Precision

In questa fase il tempo d'inferenza per ogni immagine è stato di 0.42s, per un totale di 18s per l'intero dataset di test.

6 Utilizzo ambiente ROS

6.1 configurazione

Il primo passo per la configurazione dell'ambiente di lavoro è stato quello di installare la versione di Ubuntu Xenial 16.04.1. Successivamente sono stati installati ROS e Gazebo nelle versioni precedentemente descritte.
La versione di Python utilizzata in questo progetto è la 2.7, insieme alla versione di tensorflow 2.10.
Per un corretto funzionamento è stato necessario installare le librerie all'interno del file requirement.txt, situato in detectio_door/src.

6.2 door_detection.py

door_detection.py è il file che rappresenta il nodo, che ha il compito di acquisire immagini dalla fotocamera frontale del Pepper e analizzare tali immagini per identificare gli oggetti appartenenti alle diverse classi.

```
import colorsys
import os, sys, argparse
import subprocess
import webbrowser
import time
from timeit import default_timer as timer
import tensorflow as tf
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Lambda
from PIL import Image as IMAGE
from std_msgs.msg import String
from yolo3.model import get_yolo3_model, get_yolo3_inference_model
from yolo3.postprocess_np import yolo3_postprocess_np
from common.data_utils import preprocess_image
from common.utils import get_classes, get_anchors, get_colors, draw_boxes,
    optimize_tf_gpu
from tensorflow.keras.utils import multi_gpu_model
import rospy
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import CompressedImage
import cv2
```

```
default_config = {
       "model_type": 'yolo3_efficientnet',
       "weights_path": "model.h5",
       "pruning_model": False,
       "anchors_path": os.path.join('configs', 'yolo3_anchors.txt'),
       "classes_path": "class.txt",
       "score" : 0.1,
       "iou" : 0.4,
       "model_image_size" : (416, 416),
       "elim_grid_sense": False,
       "gpu_num" : 1,
   }
```

```
class YOLO_np(object):
    _defaults = default_config

    @classmethod
    def get_defaults(cls, n):
        if n in cls._defaults:
            return cls._defaults[n]
        else:
            return "Unrecognized attribute name '" + n + "'"

    def __init__(self, **kwargs):
        super(YOLO_np, self).__init__()
        self.__dict__.update(self._defaults) # set up default values
        self.__dict__.update(kwargs) # and update with user overrides
        self.class_names = get_classes(self.classes_path)
        self.anchors = get_anchors(self.anchors_path)
        self.colors = get_colors(self.class_names)
        K.set_learning_phase(0)
        self.yolo_model = self._generate_model()

    def _generate_model(self):
        '''to generate the bounding boxes'''
        weights_path = os.path.expanduser(self.weights_path)
        assert weights_path.endswith('.h5'), 'Keras model or weights must be a .h5
            file.'

        num_anchors = len(self.anchors)
        num_classes = len(self.class_names)
        num_feature_layers = num_anchors//3

        yolo_model, _ = get_yolo3_model(self.model_type, num_feature_layers,
            num_anchors, num_classes, input_shape=self.model_image_size + (3,),
            model_pruning=self.pruning_model)

        yolo_model.load_weights(weights_path)

        yolo_model.summary()

        print('{} model, anchors, and classes loaded.'.format(weights_path))

        return yolo_model

        yolo_model.summary()

        print('{} model, anchors, and classes loaded.'.format(weights_path))

        return yolo_model

        def detect_image(self, image):
        if self.model_image_size != (None, None):
            assert self.model_image_size[0]%32 == 0, 'Multiples of 32 required'
            assert self.model_image_size[1]%32 == 0, 'Multiples of 32 required'

        image_data = preprocess_image(image, self.model_image_size)
        #origin image shape, in (height, width) format
        #image = Image.fromarray(image)
        image_shape = tuple(reversed(image.size))
```

```
        start = time.time()
        out_boxes, out_classes, out_scores = self.predict(image_data, image_shape)
        print('Found {} boxes for {}'.format(len(out_boxes), 'img'))
        end = time.time()
        print("Inference time: {:.8f}s".format(end - start))

        #draw result on input image
        image_array = np.array(image, dtype='uint8')
        image_array = draw_boxes(image_array, out_boxes, out_classes, out_scores,
            self.class_names, self.colors)

        out_classnames = [self.class_names[c] for c in out_classes]
        print("Fine detect image 1")
        return IMAGE.fromarray(image_array), out_boxes, out_classnames, out_scores,
            end , start


    def predict(self, image_data, image_shape):
        num_anchors = len(self.anchors)
        if self.model_type.startswith('scaled_yolo4_') or
            self.model_type.startswith('yolo5_'):
            # Scaled-YOLOv4 & YOLOv5 entrance, enable "elim_grid_sense" by default
            out_boxes, out_classes, out_scores =
                yolo5_postprocess_np(self.yolo_model.predict(image_data), image_shape,
                self.anchors, len(self.class_names), self.model_image_size,
                max_boxes=100, confidence=self.score, iou_threshold=self.iou,
                elim_grid_sense=True)
        elif self.model_type.startswith('yolo3_') or
            self.model_type.startswith('yolo4_') or \
             self.model_type.startswith('tiny_yolo3_') or
                 self.model_type.startswith('tiny_yolo4_'):
            # YOLOv3 & v4 entrance
            out_boxes, out_classes, out_scores =
                yolo3_postprocess_np(self.yolo_model.predict(image_data), image_shape,
                self.anchors, len(self.class_names), self.model_image_size,
                max_boxes=100, confidence=self.score, iou_threshold=self.iou,
                elim_grid_sense=self.elim_grid_sense)
        elif self.model_type.startswith('yolo2_') or
            self.model_type.startswith('tiny_yolo2_'):
            # YOLOv2 entrance
            out_boxes, out_classes, out_scores =
                yolo2_postprocess_np(self.yolo_model.predict(image_data), image_shape,
                self.anchors, len(self.class_names), self.model_image_size,
                max_boxes=100, confidence=self.score, iou_threshold=self.iou,
                elim_grid_sense=self.elim_grid_sense)
        else:
            raise ValueError('Unsupported model type')

        return out_boxes, out_classes, out_scores


    def dump_model_file(self, output_model_file):
        self.yolo_model.save(output_model_file)
```

Questo codice realizza l'implementazione del nostro modello, dove vengono caricati i pesi generati dal suo precedente addestramento. Fornisce il metodo detect_image per la predizione delle immagini di input.

```
def detect_img(yolo, image, pub):
    r_image, boxes, classes, confidences, end, start = yolo.detect_image(image)
    risultati = ""
    duration = end-start
    for i in range(0, len(boxes)):
        if(classes[i] in 'door'):
            risultato = "Porta trovata con confidenza: "+str(confidences[i])+" con
                bounding box con coordinate: x="+str(boxes[i][0])+"
                y="+str(boxes[i][1])+" larghezza="+str(boxes[i][2])+"
                altezza="+str(boxes[i][3])+" e il tempo di inferenza e' stato di
                "+str(duration)+" "
            risultati += risultato
        if(classes[i] in 'handle'):
            risultato = "Maniglia trovata con confidenza: "+str(confidences[i])+" con
                bounding box con coordinate: x="+str(boxes[i][0])+"
                y="+str(boxes[i][1])+" larghezza="+str(boxes[i][2])+"
                altezza="+str(boxes[i][3])+" e il tempo di inferenza e' stato di
                "+str(duration)+" "
            risultati += risultato
    print(risultati)
    msg = String()
    msg.data = risultati
    pub.publish(msg)
    try:
        r_image.save("risultato.jpg")
        webbrowser.open('risultato.jpg')
        print("Salvataggio riuscito")
    except:
        print('Salvataggio non riuscito')
```

La funzione detect_img richiama il metodo detect_image del modello; crea la stringa contenente le informazioni (classe, confidenza, coordinate bounding box, tempo di inferenza) relative esclusivamente alle classi door e handle, pubblicandola sul topic "/results". Infine salva il risultato sul file system in formato Jpeg.

```
def convert_depth_image(ros_image):
    cv_bridge = CvBridge()
    depth_image = cv_bridge.compressed_imgmsg_to_cv2(ros_image, 'passthrough')
    images_pepper.append(depth_image)
    cv2.imwrite("depth_img.png", depth_image)
```

Questa funzione viene usata come callback per il Subscriber. Il suo scopo è quello di acquisire l'immagine e trasformarla dal suo formato iniziale ROS in formato open cv e aggiungerla alla lista globale, per consentire la sua manipolazione futura.

```
def main(yolo, pub):
    print("Image detection mode")
    image_rob = IMAGE.fromarray(images_pepper[-1])
    detect_img(yolo, image_rob, pub)
```

La funzione main seleziona l'ultimo elemento della lista globale di immagini, la trasforma in formato PIL per consentire il corretto funzionamento del modello e infine richiama la funzione detect_img passando come parametro l'immagine in questione.

```
def pixel2depth(pub):
    rospy.init_node('detection_door',anonymous=True)
    #rospy.Subscriber("/pepper/camera/front/image_raw", Image,
        callback=convert_depth_image, queue_size=1)
    rospy.Subscriber("/pepper_robot/camera/front/image_raw/compressed",
        CompressedImage, callback=convert_depth_image, queue_size=10)
    time.sleep(3)
    while not rospy.is_shutdown():
        if len(images_pepper)>0:
            main(yolo=yolo3, pub=pub)
```

Questa funzione inizializza il subscriber e richiama iterativamente la funzione main se la lista contiene almeno un'immagine.

```
if __name__ == '__main__':
    path = os.path.dirname(os.path.abspath(__file__))
    print(path)
    os.chdir(path)
    yolo3 = YOLO_np()
    pub = rospy.Publisher("/results", String, queue_size=10)
    global images_pepper
    images_pepper = []
    pixel2depth(pub=pub)
```

Nell'ultima parte del codice, andiamo ad individuare il path relativo al nodo, inizializziamo il modello e il Publisher, definiamo la lista di immagini globale e lanciamo la funzione "pixel2depth".

6.3 Testing su file .bag

Tramite camera frontale del robot, sono state acquisite immagini relative ad un percorso all'interno del dipartimento, successivamente raccolte all'interno di un file bag, sul quale è stato eseguito un ulteriore test.

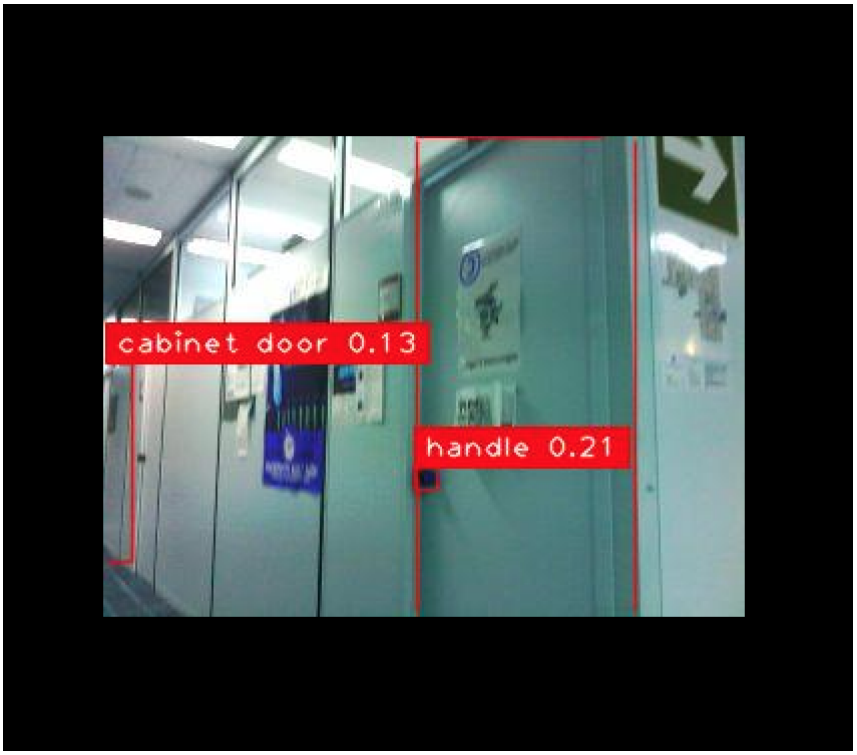

Figura 24. Foto file bag 1

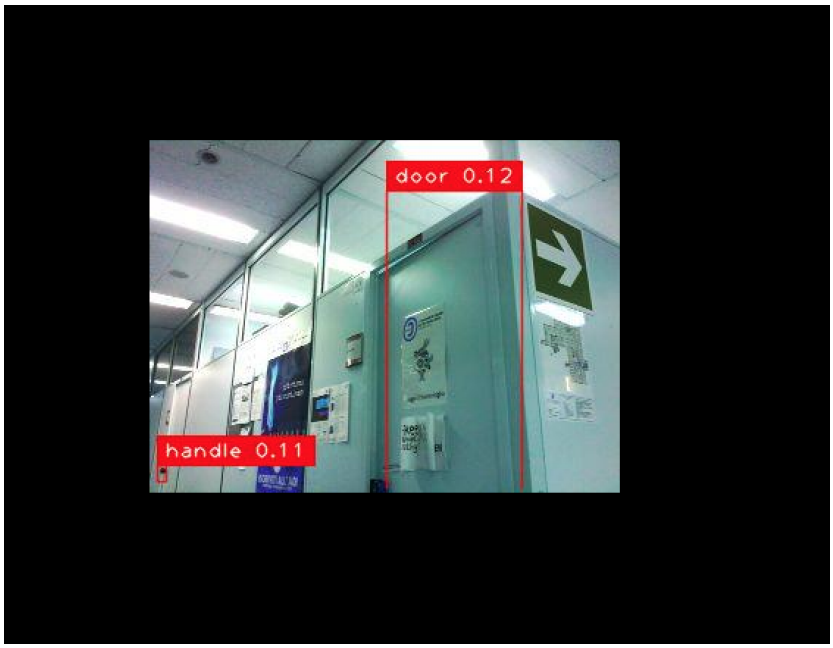

Figura 25. Foto file bag 2

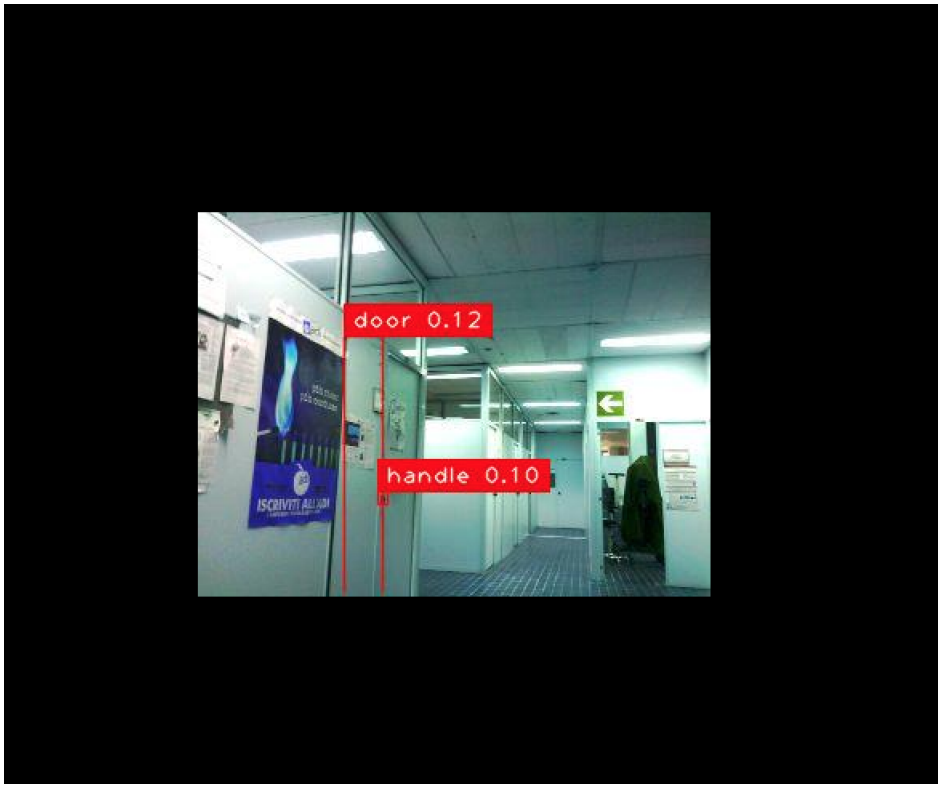

Figura 26. Foto file bag 3

Figura 27. Foto file bag 4

Dalle immagini possiamo osservare che il rilevamento delle classi avviene con successo, riscontrando una porta e una maniglia dove effettivamente sono presenti. Dal test completo si evince che se l'oggetto da analizzare non si trova di fronte al robot, la confidenza degli oggetti analizzati è molto bassa. La confidenza aumenta significativamente quando l'oggetto è posizionato perfettamente davanti alla camera. Essendo i corridoi del dipartimento di colore totalmente bianco, distinguere le porte non risulta essere un compito semplice, nonostante questo la rete produce risultati soddisfacenti, identificando, nella maggior parte dei casi, l'oggetto correttamente.

6.4 Topic

Come già scritto precedentemente nella descrizione relativa al codice del nodo, all'interno del topic sono state inserite le informazioni riguardanti porte e maniglie (classe, confidenza, coordinate bounding box, tempo di inferenza).

```
daniele@ubuntu:~$ rostopic echo /results
WARNING: no messages received and simulated time is active.
Is /clock being published?
data: "Maniglia trovata con confidenza: 0.27891407041186156 con bounding box con coordinate:\
  \ x=7 y=25 larghezza=59 altezza=53 e il tempo di inferenza e' stato di 7.15308690071\
  \       "
---
data: "Porta trovata con confidenza: 0.9555806135739289 con bounding box con coordinate:\
  \ x=128 y=16 larghezza=252 altezza=229 e il tempo di inferenza e' stato di 6.55046916008\
  \     Maniglia trovata con confidenza: 0.6047580486296198 con bounding box con coordinate:\
  \ x=226 y=131 larghezza=238 altezza=145 e il tempo di inferenza e' stato di 6.55046916008\
  \      Porta trovata con confidenza: 0.15216316510403516 con bounding box con coordinate:\
  \ x=85 y=43 larghezza=122 altezza=187 e il tempo di inferenza e' stato di 6.55046916008\
  \      "
---
data: "Porta trovata con confidenza: 0.9485438187745814 con bounding box con coordinate:\
  \ x=129 y=15 larghezza=252 altezza=229 e il tempo di inferenza e' stato di 5.85708093643\
  \     Maniglia trovata con confidenza: 0.5499582651141282 con bounding box con coordinate:\
  \ x=226 y=131 larghezza=238 altezza=145 e il tempo di inferenza e' stato di 5.85708093643\
  \       "
---
data: "Porta trovata con confidenza: 0.9485438187745814 con bounding box con coordinate:\
  \ x=129 y=15 larghezza=252 altezza=229 e il tempo di inferenza e' stato di 5.87337684631\
  \     Maniglia trovata con confidenza: 0.5499582651141282 con bounding box con coordinate:\
  \ x=226 y=131 larghezza=238 altezza=145 e il tempo di inferenza e' stato di 5.87337684631\
  \       "
---
data: "Porta trovata con confidenza: 0.9485438187745814 con bounding box con coordinate:\
  \ x=129 y=15 larghezza=252 altezza=229 e il tempo di inferenza e' stato di 5.71653985977\
  \     Maniglia trovata con confidenza: 0.5499582651141282 con bounding box con coordinate:\
  \ x=226 y=131 larghezza=238 altezza=145 e il tempo di inferenza e' stato di 5.71653985977\
  \       "
---
```

Figura 28. Risultati topic

6.5 Istruzioni avvio

Per avviare il nodo è sufficiente seguire i seguenti passaggi:

1. Avvio della simulazione tramite comando: roslaunch PackageName SimulationName.launch
2. Avvio del nodo tramite comando: rosrun detection_door detection_door.py
3. Avvio del topic tramite il comando: rostopic echo /results

7 Conclusioni

Uno dei problemi principali di qualunque robot è l'interazione con l'ambiente circostante, nel nostro caso il Pepper si muove all'interno di un edificio, memorizzando al suo interno la mappa del dipartimento, sapendo quindi la posizione dei varchi che collegano le stanze. Tramite la realizzazione del nostro nodo ora Pepper ha la possibilità di individuare la presenza di porte in corrispondenza di tali varchi.
Si potrebbe realizzare un nuovo nodo ROS, tramite il quale permettere al Pepper, a seguito di un'identificazione, di interagire con l'oggetto maniglia, al fine di permettere l'apertura dell'eventuale porta.
In una futura implementazione, per migliorare le performance del nodo, è possibile tentare l'utilizzo di altre reti adatte al nostro scopo, tra le quali troviamo SiamMask, Deep SORT, Tracktor++.
Per una più semplice condivisione del progetto e per permettere a chiunque di poter ampliare il progetto apportando migliorie, abbiamo caricato il package all'interno di un repository github: https://github.com/MiguelARD/DoorDetect-Dataset.