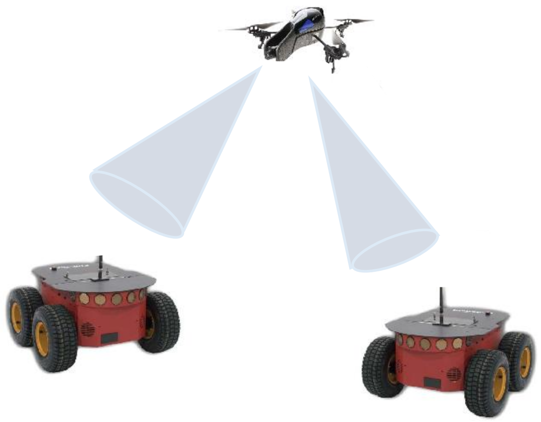
PROJECT : MUROPHEN



Deep Learning based Target Tracking & Occlusion Avoidance for Aerial Robots

V.Kodipaka[1], M.Jacquet[2], D.Sidobre[3], A.Franchi[2,3]

LAAS-CNRS, Thesis Internship

MS in Computer Vision, University of Burgundy, France

(6th July 2020)

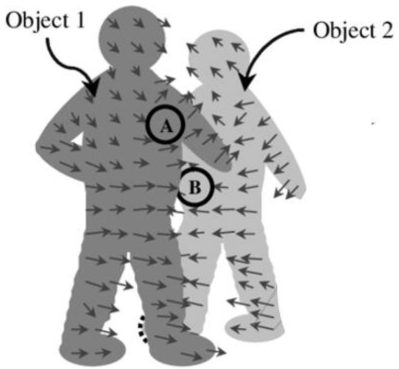
Overview

- Introduction
- Methodology
- Time to occlusion
- Computing tocc
- Meaning of DL for UAV
- Choice of Yolo and Retinanet
- YOLO & Retinanet Architectures
- Implementation
- Results and performance
- Processing of Depth
- Conclusion
- Future work
- Acknowledgments

Introduction

- **Motivation : uav+occlusion+cnn+tracking**

why drones: surveillance, HCI, navigation, trajectory planning etc
why tracking (vision): study object's behavior, motion detection
why occlusion: spatial-temporal overlap, periodic occurrence etc
why CNN: improve accuracy of detection vs computational cost



Partial occlusion

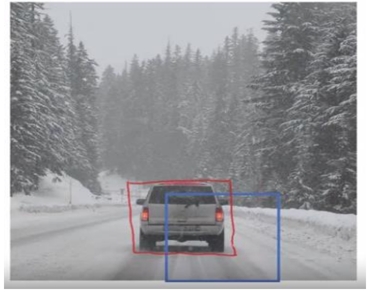
- **explanation of terms**

Occlusion: inability to track the features when obstacle interferes CFOV & MT

CNN: NN with set of Conv layers (follows Back Prop, Gradient Descent for SL)

Object detection: R/CwL + bounding box for the object in an image

Efficient computations / outputs inaccurate position of BB

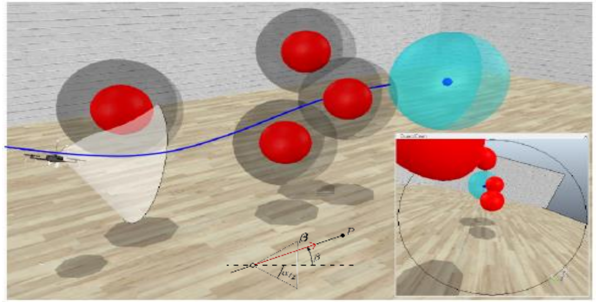


CNN SS – PDW for Obj Detection

Methodology

Background

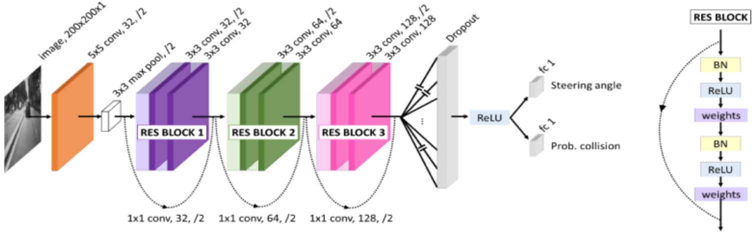
- Robust occlusion handling mechanism (2007)
- Anti-occlusion observation model & auto-recovery (2016)
- Hybrid occlusion free tracking (2017)
- Vision-based reactive planning (2018)
- ETH’s Dronet with Udacity & Collision dataset (2018)



Aggressive target tracking avoiding occlusions

Motivation:

for the use of dnn over occlusion free tracking

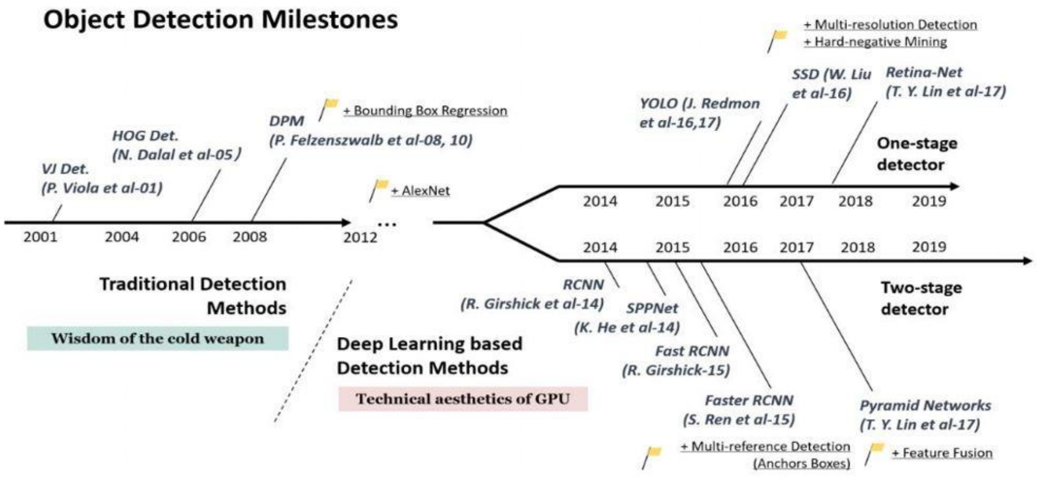


DNN based collision free navigation

Methodology

State of the art : DL

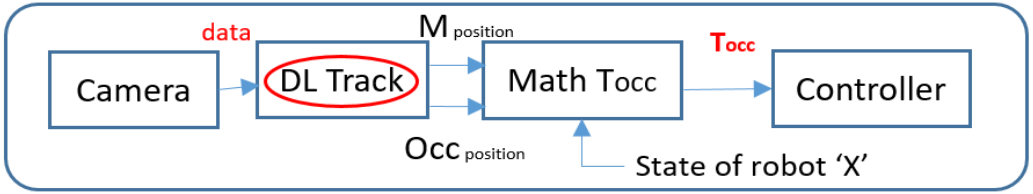
- Single stage
- Two stage
- Data as points
- Data Augmentation



Feature Pyramids, Region proposals, Proximal Policy, encoder-decoder type, Point cloud type, Pooling boosting, 3D voxels, Memory eff. based, Gaussian occlusion, Contrastive Learning etc.

Over 20 latest DNN papers were referred and tableted for performance analysis

Proposed Control Scheme



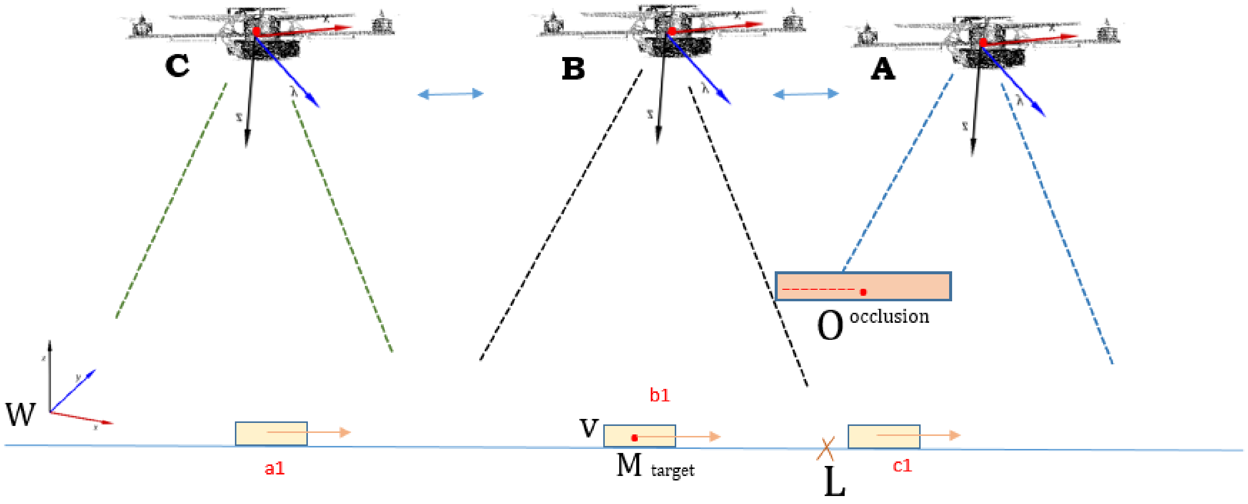
Custom Object Detection in SIX STEPS:

- Creating own Dataset
- Drawing Annotations
- Downloading Weights
- Training phase
- Export Inference Graph
- Testing Phase

From target position and occlusion positions we calculate the tocc (if the state of the robot is known)

Thus computed Tocc is fed to controller for the next possible move

Time to Occlusion



The time instant at which object is going to be occluded is termed to be *"Time of Occlusion"*

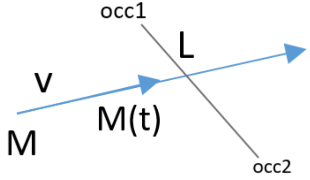
Time difference of current instance of target to the time instance at the point of occlusion - 'Time to Occlusion'

This differential time help the controller to command a move in a proper direction to avoid occlusion

Time of Occlusion : Case1

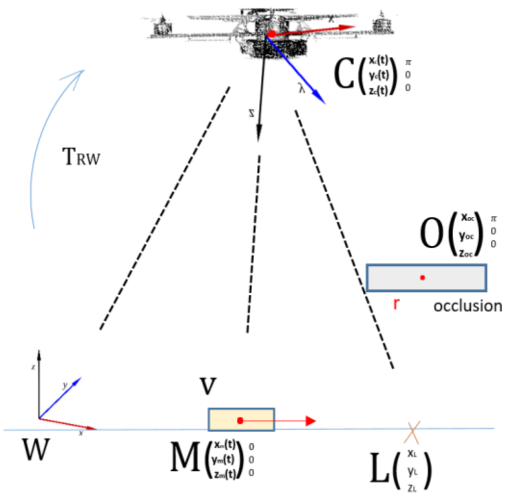
Let ${}^{W}C = \begin{bmatrix} x_C & y_C & z_C \end{bmatrix}_W^{\top}$, ${}^{W}\eta_C = \begin{bmatrix} \phi_C & \theta_C & \psi_C \end{bmatrix}_W^{\top}$ and ${}^{W}R_C$ are the position, orientation and rotation

$$t_{Occ} = \frac{\|M - L\|}{v_M}$$



For straight line trajectory (linear) with constant velocity

- assumptions: linear motion, shadow of occlusion on the ground
- to find: intersection L with the line, distance vs velocity
- method: intersection L is the scaled version of occlusion point
- our result: final formula



target considered has zero size & is on the ground:

$$
\begin{aligned}
{}^{W}z_L &= 0 \\
{}^{C}L &= \lambda \times {}^{C}\overline{Occ} \\
{}^{W}L &= {}^{W}t_C + {}^{W}R_C {}^{C}L \\
&= {}^{W}C + {}^{W}R_C {}^{C}L \\
&= {}^{W}C + \lambda {}^{W}R_C {}^{C}\overline{Occ}
\end{aligned}
$$

$$0 = {}^{W}z_C + \lambda \begin{bmatrix} -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \begin{bmatrix} x_{Occ} - r \\ y_{Occ} - r \\ z_{Occ} \end{bmatrix}_C$$

$$0 = {}^{W}z_C + \lambda \begin{bmatrix} -s\theta(x_{Occ} - r) + s\phi c\theta(y_{Occ} - r) + c\phi c\theta z_{Occ} \end{bmatrix}$$

$$\boxed{t_{Occ} = \frac{\|M - \lambda \overline{Occ}\|}{v_M}}$$

$$\text{where } \lambda = \frac{{}^{W}z_C}{s\theta(x_{Occ} - r) - s\phi c\theta(y_{Occ} - r) - c\phi c\theta z_{Occ}}$$

Knowns:

$${}^{W}C, {}^{W}\eta_C$$

$${}^{C}M, {}^{C}\dot{M} = {}^{C}v_M, {}^{C}\ddot{M} = 0$$

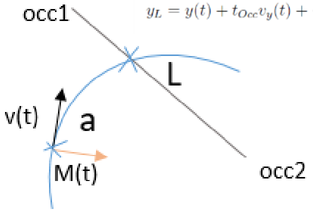
$${}^{C}\overline{Occ} = \begin{bmatrix} x_{Occ} - r \\ y_{Occ} - r \\ z_{Occ} \end{bmatrix}_C$$

$$L = \begin{bmatrix} x_L \\ y_L \\ z_L = 0 \end{bmatrix}$$

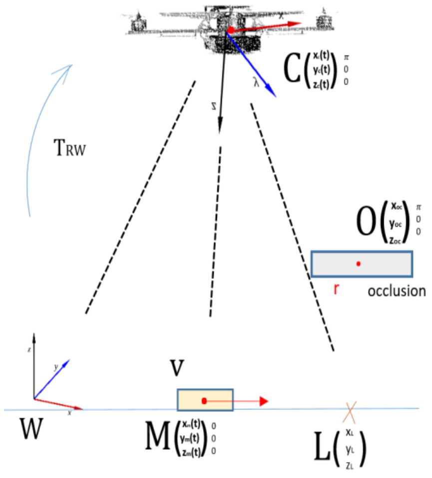
Time of Occlusion : Case2

Let ${}^{W}C = \left[x_C \; y_C \; z_C\right]_W^{\top}$, ${}^{W}\eta_C = \left[\phi_C \; \theta_C \; \psi_C\right]_W^{\top}$ and ${}^{W}R_C$ are the position, orientation and rotation

$$0 = x_L + a y_L + b$$
$$x_L = x(t) + t_{Occ} v_x(t) + \frac{1}{2} t_{Occ}^2 a_x$$
$$y_L = y(t) + t_{Occ} v_y(t) + \frac{1}{2} t_{Occ}^2 a_y$$



Moving target with time-varying speed with constant acceleration



- assumptions: 2nd order trajectory, linear occlusion, on the ground
- to find: intersection L with the line, we have two parameters a and b
- method: write the parametric equation, equal them in point L, solve
- our result: final formula

$$\boxed{t_{occ} = \frac{-[v_x(t) + a v_y(t)] + sqrt([v_x(t) + a v_y(t)]^2 \pm 2[(a_x + a a_y)][x(t) + a y(t) + b])}{(a_x + a a_y)}}$$

For occl. point-1: ${}^{W}L1 = {}^{W}t_C + {}^{W}R_C {}^{C}L1$

For occl. point-2: repeat same as above, we get occlusion point-2

From 2 points eqn form: $[y - y_{occ1}] = \frac{[y_{occ2} - y_{occ1}]}{[x_{occ2} - x_{occ1}]} \times [x - x_{occ1}]$

$$x - y\left[\frac{\gamma_1 - \gamma_1^*}{\gamma_2 - \gamma_2^*}\right] + \left[\frac{{}^{w}yc(\gamma_1 - \gamma_1^*) + {}^{w}xc(\gamma_2^* - \gamma_2) + {}^{w}zc(\gamma_1^*\gamma_2 - \gamma_1\gamma_2^*)}{(\gamma_2 - \gamma_2^*)}\right] = 0$$

$$\boxed{a = \left[\frac{x_{occ1} - x_{occ2}}{y_{occ2} - y_{occ1}}\right]}$$

$$\boxed{b = \left[\frac{{}^{w}xc(y_{occ2} - y_{occ1}) + {}^{w}yc(x_{occ1} - x_{occ2}) + {}^{w}zc(x_{occ2}y_{occ2} - x_{occ2}y_{occ1})}{(y_{occ2} - y_{occ1})}\right]}$$

Knowns:

$${}^{W}C, {}^{W}\eta_C$$
$${}^{C}M(t), {}^{C}\dot{M}(t) = {}^{C}v(t), {}^{C}\ddot{M} = a$$
$${}^{C}\overline{Occ} = \begin{bmatrix} x_{Occ} - r \\ y_{Occ} - r \\ z_{Occ} \end{bmatrix}_C \qquad L = \begin{bmatrix} x_L \\ y_L \\ z_L = 0 \end{bmatrix}$$

$$\beta_1 = \left[c\theta c\psi(x_{Occ} - r) + (-c\phi s\psi + s\phi c\psi s\theta)(y_{Occ1} - r) + (s\phi s\psi + c\phi s\theta c\psi)z_{Occ1}\right]$$
$$\beta_2 = \left[c\theta s\psi(x_{Occ1} - r) + (c\phi c\psi + s\phi s\psi s\theta)(y_{Occ1} - r) + (s\phi c\phi)z_{Occ1}\right]$$
$$\alpha = \left[-s\theta(x_{Occ1} - r) + (s\phi c\theta)(y_{Occ1} - r) + (c\phi c\theta)z_{Occ1}\right]$$

$$\gamma_1 = \beta_1/\alpha, \gamma_2 = \beta_2/\alpha$$
$$\gamma_1^* = \beta_1^*/\alpha^*, \gamma_2^* = \beta_2^*/\alpha^*$$

Deep Learning for UAV

Why is it effective?

- abilities of DL to learn from raw sensor data
- higher-level abstractions - complex behaviors for supervised learning
- lower levels of abstraction - feature extraction - computational resources
- online processing, limiting the applications - reactive behaviors
- tuning the network hyper-parameters is easy

Constraints

- Processors with higher configurations
- GPUs for Parallel Computing
- Suitable libraries for installations

Choose YOLO/RetinaNet

YOLO

- Outputs precise bounding boxes
- Pixel level / finer grid – object/BB assignment
- Faster detection rates – Batches/epochs
- Multiple classes prediction

RetinaNet

- Extreme foreground-background scenarios
- Loss estimation class imbalances
- Irregular object shapes

Drawbacks:

YOLO – 2 objects associates with same grid cell
& both have same anchor box shape

YOLO – 2 anchor boxes but 3 objects in the grid cell

RetinaNet – pixel level computation and class assignment

RetinaNet – accurate but lesser faster compared to YOLO

YOLO

Joseph Redmon – Father of YOLO

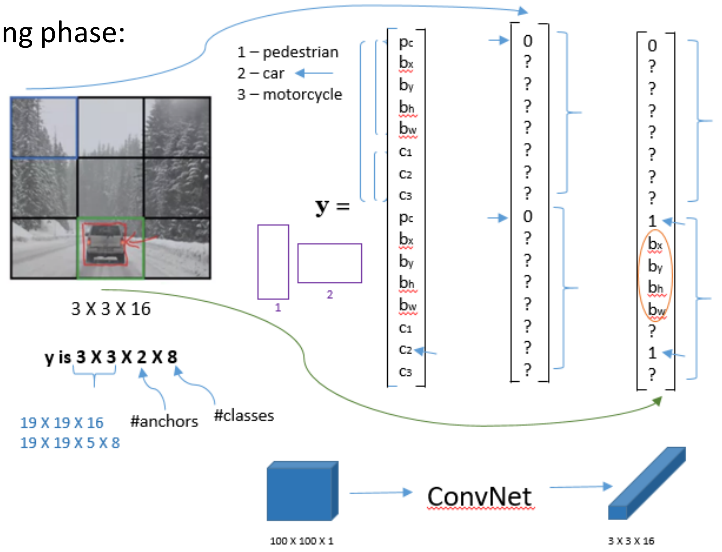
YOLO versions

- YOLOv1 (E2016) - 24 conv layers with 2 FC Layers
- YOLOv2 (L2016) - 19 layers + 11 more layers
- YOLOv3 (2018) – Darknet-53 + 53 detection layers (45.1AP)
- YOLOv4 (Apr 2020) – advanced features from *Alexey B.*

Key aspects of YOLO:

- Mapping output volume to input image
- 8D Vector – Pc, Bound box, object classes
- IOU (Intersection over Union) : B box
- Anchor boxes – multiple class detection
- Non-max suppression – threshold set for multiple classes

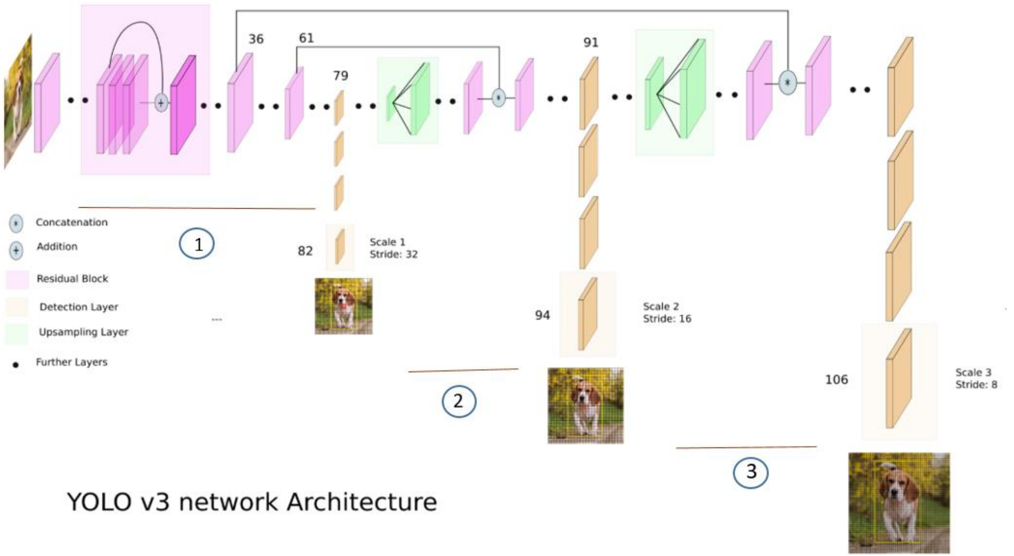
Training phase:



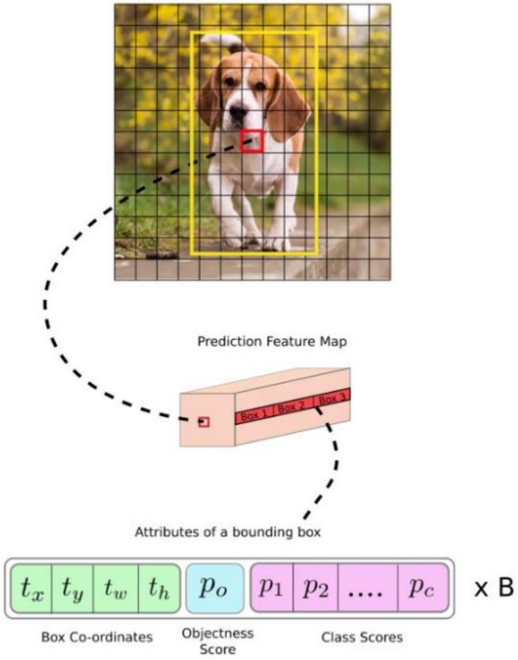
Testing phase:

After Predictions from Training, Non-max suppressed output

YOLO Architecture

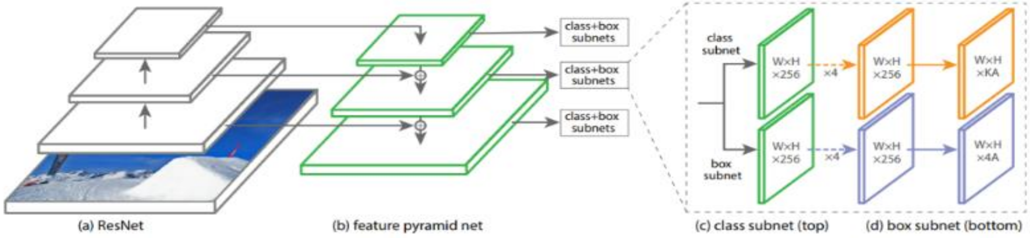


YOLO v3 network Architecture

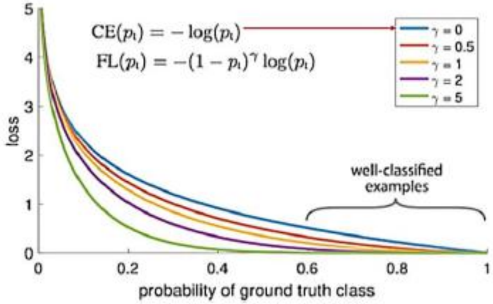


Stride: ratio by which it down-samples the input
Scale: size of image output for accuracy in Detection of the object classes
Kernel size: generates Feature map

YOLOv4: YOLOv3 + Weighted-Residual-Connections, Cross-Stage-Partial-Connections, Cross mini-Batch Normalization, Self-adversarial-training, Mish-activation, Mosaic data augmentation, Drop Block regularization, CIoU loss features



RetinaNet Architecture

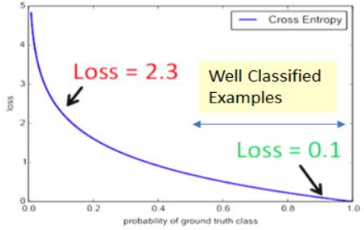


$$\boxed{\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_i)}$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases}$$

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t).$$

- 100000 easy : 100 hard examples
- **40x bigger loss** from easy examples



- **Bottom-up pathway:** ResNet – cal. feature maps at different scales for input image size
- **Top-down pathway:** up-samples spatially feature maps from higher pyramid levels (FPN)
- **Lateral connections:** merge the layers with same spatial size.
- **Classification subnet:** predicts the probability of object at each spatial location
- **Regression subnet:** regresses the offset for the bounding boxes from anchor boxes (GTO)

Detectron2 : Detectron + panoptic segmentation, dense pose, Cascade R-CNN, rotated bounding boxes, etc

YOLO Implementation

YOLOv3:

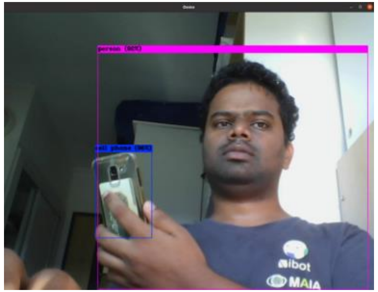
- Transfer Learning – pre-trained net, pretrained weights, predef-annotations
- TensorFlow 1.15, Keras 2.2.4, Numpy 1.16, Opencv 3.6.9,python 3.2.4.17
- batch size: 8 (can be 8 or 32). Batch Norm Epsilon= 1e-5, leaky ReLU=0.1, anchor boxes - 9 no.s, epoches: CPU
- Backbone: Darknet-53 residual connections, 72 Conv. Layers, Upsample layer, Non-max suppression
- COCO Dataset: 80 classes

YOLOv4:

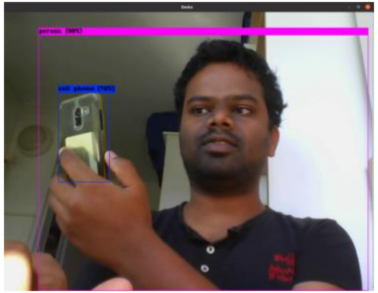
- Opencv>=2.4, intel i7, cmake>=3.8, openmp, git, g++,
- darknet backbone, for GPU usage: CUDA 10.0, CuDNN >=7.0

YOLO outputs using Webcam

v3 CPU



v4 CPU



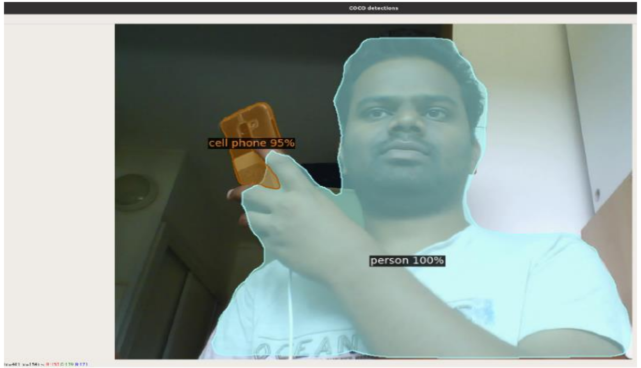
v4 GPU

RetinaNet Implementation

Detectron2:

- RetinaNet includes mask RCNN
- Written in Python - powered by Pytorch
- Uses COCO Dataset
- Multiple Networks grouped
- python>=3.6, Pytorch>=1.4, torchvision, pycocotools,
- opencv-python, opencv>=4, gcc>=5, CUDA 10.1, CuDNN >=7.0

Detectron2 - Outputs using Webcam



CPU usage

GPU usage

Performance:

Algorithm	CPU or GPU	Efficiency
YOLOv3	Intel® Core^{TM} i5-4210U CPU(Ubuntu18.04)	40.1mAP / @2 fps
YOLOv3	Intel® Core^{TM} i7-7820U CPU(Ubuntu16.04)	41.4mAP / @6 fps
YOLOv4	Intel® Core^{TM} i7-7820HQ CPU(Ubuntu20.04)	40.1mAP / @0.2 fps
YOLOv4	Intel® Core^{TM} i7-7820HQ GPU(Ubuntu20.04)	42mAP / @4.8 fps
Detectron2	Intel® Core^{TM} i7-7820HQ CPU(Ubuntu20.04)	2.35s/iter
Detectron2	Intel® Core^{TM} i7-7820HQ GPU(Ubuntu20.04)	1.93s/iter

The higher the values of mAP and fps the better the performance of tracking system

- In short, to detect the distance of target from camera, triangle similarity technique can be used.

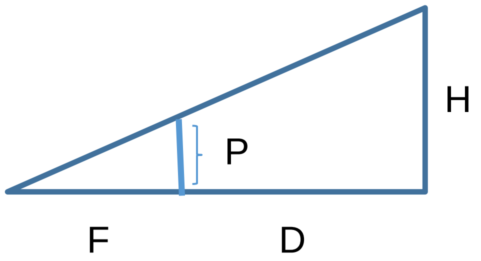
- Let a target is at a distance D (in cms) from camera and the target's actual height is H
- Using the object detection, we identify the pixel height P of the person using the bounding box coordinates
- Using these values, the focal length of the camera can be calculated:

Eq 1: F = (P x D) / H

- After calculating the focal length of the camera, we can use the actual height H, P and F to calculate the distance (depth) of the target from camera.
- Distance from camera can be calculated using:

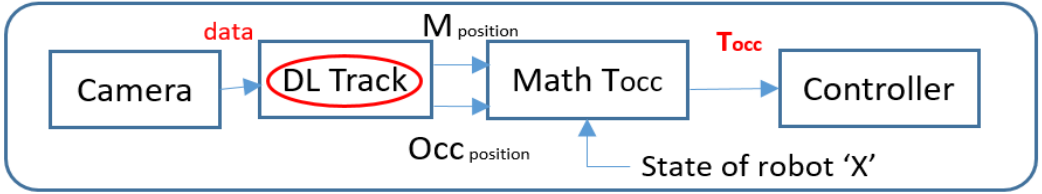
Eq 2: D' = (H x F) / P

- Repeat this for finding depth of occlusion for every frame
- we can move on to calculate the distance (Euclidean dist) between the objects



***concept developed by external sources social distancing during COVID crisis**

Conclusion & Future Work



Downward looking camera in a 3D world:

- math model for time of occlusion
- tracking by YOLO
- tracking by RetinaNet
- Mpos, Opos are used to compute Tocc
- computed **Tocc** fed to controller

*** COVID-19 situation jeopardized the work**

Inspired from:

Joint Monocular 3D Detection Vehicle Detection and Tracking
– tuning YOLO/RetinaNet to Depth based tracking system

Further to investigate:

- Depth Estimation based tracking: triangular similarity,
  "Real time Depth Estimation and Obstacle Detection from Monocular Video"
- Simulations on Robotic Platform – Math Model + DL Algor.

Thanks to:

Mr. Martin Jacquet (PhD with Antonio),
Prof. Antonio Franchi (LAAS-CNRS),
Team of RIS-LAAS CNRS
Free Coursera online (DL & AR)
paperswithcode.com

THANK YOU!!!