Object-detection

Smart warehouse: pakketjes ophalen en verplaatsen.. Kiezen welke robot doet wat, rekening houden welke robot is het dichtste bij? Intelligente keuze maken welke robot wat moet doen, alsook hoe hij dat moet doen

Object en people tracking als service on the rest

Stroom afbeeldingen, afbeeldingen classificeren

Niet enkel werken voor afb, ook objecten

Image segmentatie

Voorspelling naar waar objecten gaan

Camera/LiDar/Sonar…

Taken geven verschillende soorten output

Gedistribueerd systeem -> software draaien op laptop/jetson…

Image classification: in cloud runnen of op embedded device (jetson nano die 5W) trekt

Software proper inpakken, gemakkelijk distribueren -> docker

Requirements: niet teveel veranderen: drie netwerken in parallel terwijl origineel maar één netwerk..

Planning: Embedded AI

Python OpenCV:

- camera images: check

- Lidar data: <https://medium.com/@peterfitch/opencv-for-semantic-segmentation-of-drains-from-lidar-775cf5db1d82>

- Intermediate featuremaps?

Network in twee kappen, output in midden

- Video feeds -> check!

-

Krijgen we informatie meegegeven? Zelf detectie welke input te gebruiken? -> alle informatie op voorhand verkregen, weten op moment wanneer systeem opstart

Generic systeem, toekomstig RL agents, graph convolutie… niet allemaal implementeren, gemakkelijk nieuwe types van inputs kunnen toevoegen

Nieuw Neuraal Net toevoegen om bepaalde input te gebruiken

Outputs: informatie samenvoegen, ookal van ander type net -> soort van unified formaat voor vinden, bv hond of kat, ook object detectie: boundingboxes, image segmentation.. zelfde verhaal: gemakkelijk extra output types ondersteunen! Dynamisme in systeem

Inputs krijgen: krijgen we string? File formaat? Voornaamste input: team dat robots bestuurt olv Thomas Huybrechts of input van de simulatie

Outputs: mannen distributed worldmaps -> duidelijke afspraken maken

HTTP POST method, afspreken hoe interfaces eruit zien en hoe bereikbaar

Python mag gebruikt worden, niet te exotisch

1 generiek software systeem

1 stuk software: omgaan met objectdetectie, …

Modellen: voorgetrainde modellen: zeker met ONNX overweg kunnen!

Pytorch: state dictories.. leuke extras, ONNX belangrijkste voor het systeem

Zelf implementatie schrijven voor lidar en point clouds

Gemakkelijk juiste klasses etc kunnen aanpassen om bv RL agents in te kunnen steken

Camera: niet altijd RGB, ook RGB-D -> diepte toegevoegd

High level diagram: high-level architectuur uitwerken

Soorten objecten? # aantal verschillende objecten?

Baseren op bestaande nets -> CIFAR10… voorkeur ImageNet

Coco Kitti voor objectdetectie

Deze datasets moeten kunnen worden ondersteund: MODULARITEIT, FLEXIBEL

Video feeds -> verschil met foto’s

Videostream: elk frame apart behandelen: KMAN filter klassiek, ML voor tracking van objecten

Zowel classificatie : deze afbeelding: wat is dit? Systeem zegt wat het is

Objectdetectie: waar in deze afbeelding zijn er objecten en waar zijn deze? Mens.. rechthoekje rond ding tekenen

Allebei apart kunnen uitvoeren

Voor elk soort data netwerk trainen: voor elk soort data, ander soorten modellen trainen… BV LIDAR/SONAR

Off the shelf modellen gebruiken! Meesten al in Pytorch te downloaden

Netwerk getraind op afbeeldingen: niet werken op lidar data, systeem weet welk soort we krijgen: systeem moet met idee om kunnen, we krijgen juist getrainde netwerken

Input RGB afbeelding: grootte ook van belang -> corrigatie kunnen uitvoeren door systeem zodanig dat het past op netwerk

APlicatie werkend waar afbeelding erin komt, werkende classificatie, objectdectie.. voor verschillende taken kunnen uitvoeren

Data op verschillende manieren binnen: constante stream RGB afbeeldingen, andere kant request van afbeelding, voer classificatie op -> verificatie van bv plaats robot in het warehouse

Stream van foto’s die we binnenkrijgen: communiceren met andere teams

Communicatie over DUST

PyTorch blijft sowieso, opensource standard ONXX, speciefiek op Tensor: om gebruikt model te verplaatsen naar ander model -> zeker ondersteunen

OpenCV: over nadenken, objectdetectie stukken mss niet van gebruiken

ONXX: C++ gaan gebruiken om runner van te maken

Docker vereiste voor distributed inference

Mogelijke inputtypes: camera images, LIDAR, intermediate featuremaps -> ruwe tensors doorheen model: doorsturen, in principe model in twee opsplitsen

Feature detector bijvoorbeeld opsplitsen: zwaardere berekeningen op jetson, andere op cpu..

Point cloud segmentatie: interessant

General classification en general regression: optimaal als we deze allemaal ondersteunen maar beginnen met images en point clouds

Runners gaan maken voor Jetson (in C++) alsook coral!

Docker: specifieke runners uitvoeren specifieke code.. Python en C++ door elkaar OK

C++ meer voor distributed inference, beter sneller.. dan Python

DUST communicatie tussen blokken, data standaard in zelfde formaat…

ONXX kijken, hoe Pytorch er mee omgaat

ONXX: Open Neural Network Exhange: open format om machine learning modellen voor te stellen

Pytorch