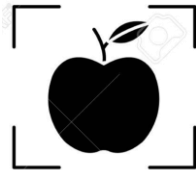


UNIVERSITÀ DEGLI STUDI DI SALERNO

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED
ELETTRICA E MATEMATICA APPLICATA**

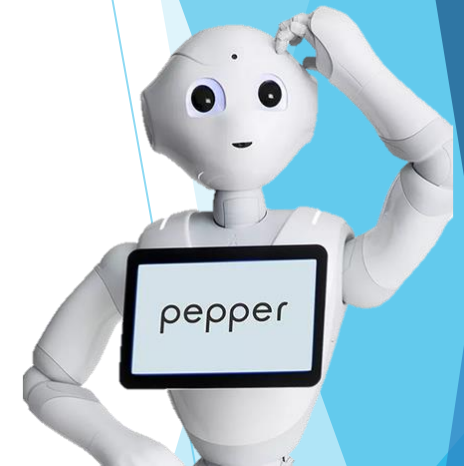


COGNITIVE ROBOTICS PROJECT

PEPPER SOCIALE



Group 18



GIOVANNI PUZO

VITTORIO FINA

VINCENZO RUSSOMANNO

SALVATORE VENTRE

ANNO ACCADEMICO 2020-2021



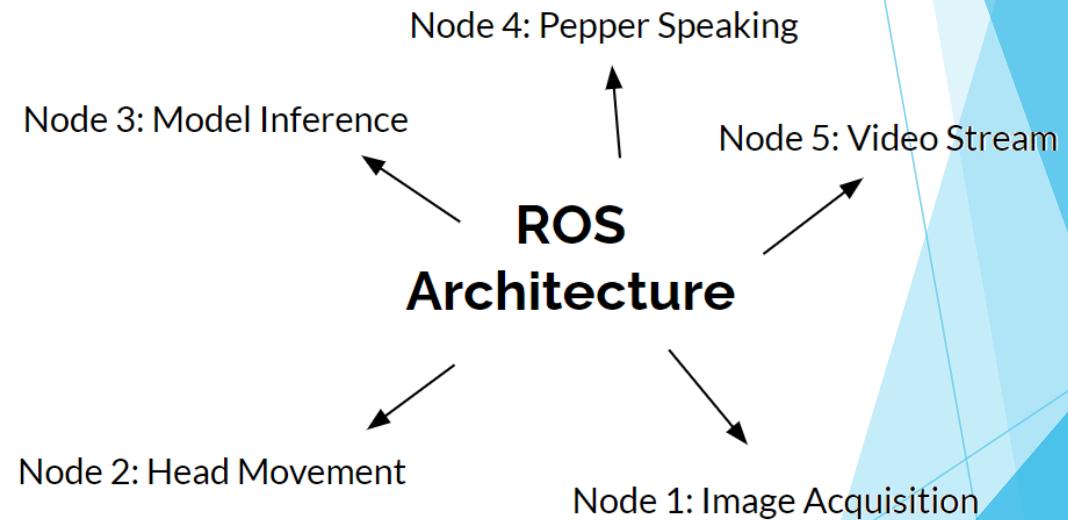
.DIEM

ARCHITETTURA (1)

L'architettura sviluppata prevede l'utilizzo di 4 packages fondamentali, in cui sono raggruppati i nodi da noi sviluppati, alcuni dei quali cooperano usando i topic e la tecnica di comunicazione publisher/subscriber ed altri tramite i servizi, per permettere a Pepper di eseguire un dato compito.

Possiamo raggruppare i nodi come segue:

- Image Acquisition
- Head Movement
- Animated Say
- Object Detection
- Video Stream



ARCHITETTURA (2)

In particolare:

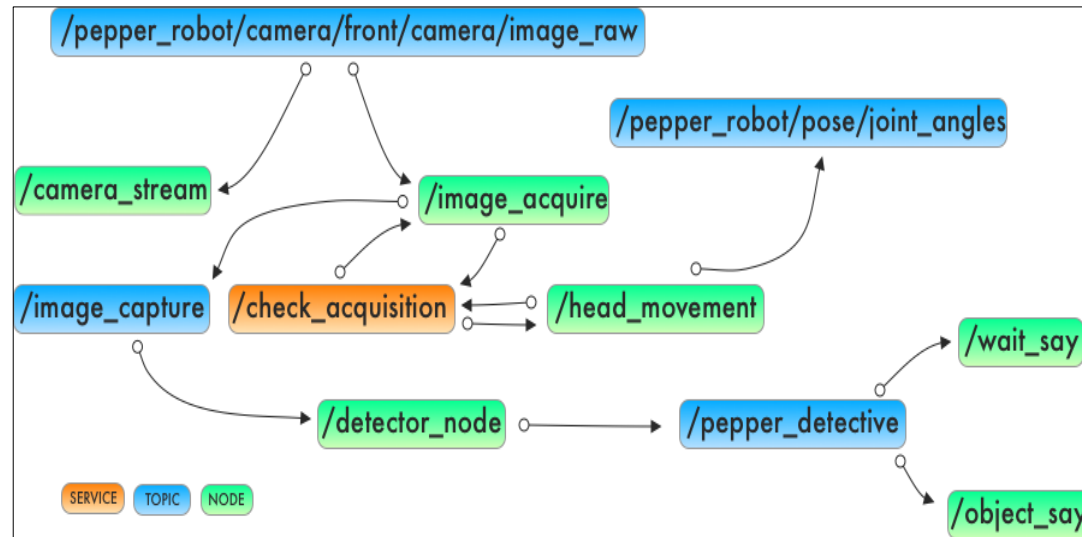
1. **Camera Acquisition:** ha l'obiettivo di gestire gli aspetti relativi alla fotocamera frontale di Pepper, che si tratti di acquisizione di immagini o di mostrare il flusso video.
2. **Head Movement:** ha l'obiettivo di gestire gli aspetti relativi ai movimenti della testa di Pepper e comunicare con il topic responsabile del movimento dei giunti per far sì che Pepper possa ruotare la testa nelle posizione desiderate.
3. **Animated Say:** ha l'obiettivo di gestire gli aspetti relativi alle capacità di Pepper di parlare.
4. **Object Detection:** ha l'obiettivo di gestire gli aspetti relativi alle funzionalità di Pepper per il rilevamento degli oggetti in una scena. Una descrizione dettagliata del detector utilizzato sarà mostrata nelle diapositive successive.





ARCHITETTURA (3)

Descrizione dettagliata dei protocolli di comunicazione utilizzati per far interagire i nodi creati, compresi i meccanismi di *publisher/subscriber* coi relativi topics e l'architettura dei servizi utilizzata:



Nella foto si possono vedere i nodi in esecuzione in *verde*, gli argomenti che permettono interazioni in *blu* e i servizi disponibili in *arancione*. Una freccia che va da un **topic** ad un **nodo** indica che quel nodo è inizializzato come *subscriber*, nel caso contrario come *publisher*; per i *servizi* abbiamo una freccia bidirezionale.





SCELTE IMPLEMENTATIVE (1)

Lo sviluppo del progetto è stato caratterizzato da alcuni aspetti significativi:

- **Uso di msg files: `ImageWithPose.msg`, `DetectionWithPose.msg`**
sono semplici files di testo che descrivono i campi di un messaggio ROS e sono memorizzati nella directory *msg* di un package:
 - ***ImageWithPose.msg***: Position (string) + Image
 - ***DetectionWithPose.msg***: Position (string) + Detections (vector)
- **Launch file**: tutti i nodi vengono lanciati contemporaneamente col file *pepper.launch* e viene eseguito il flusso logico del sistema, dal movimento della testa all'inferenza sulle immagini catturate.
- **Startup head position check**: Prima dell'acquisizione delle immagini, viene effettuata una fase di configurazione per far sì che Pepper si trovi in una giusta posizione di partenza.





SCELTE IMPLEMENTATIVE (2)

Altri aspetti significativi relativi allo sviluppo del progetto sono:

- Uso dei **services**: per la gestione della cattura delle immagini in risposta alla rotazione della testa del robot.
 - **Client** Head Movement Node: si occupa di preparare l'acquisizione dell'immagine della camera per una certa «head position».
 - **Server** Camera Acquisition: ha il compito di recuperare il frame corrente dalla camera e pubblicarlo sul topic `/image_capture`.

Questo meccanismo risulta utile perché il nodo `/head_movement` ha come unica preoccupazione quella di pubblicare messaggi sul topic relativo al movimento dei giunti e, una volta in posizione, chiamare il servizio appena citato per far catturare l'immagine nella posizione desiderata, ricevendo risposta negativa o positiva.





MODELLO (1)

È stato scelto di utilizzare uno dei modelli pre-addestrati sul dataset COCO 2017, in particolare quello più performante considerando un compromesso tra il tempo di elaborazione e mAP. Un'altra caratteristica tenuta in considerazione è stata la dimensione del modello, in modo tale da avere un tempo ragionevole di caricamento nel robot.

Model	test-dev			val	Params
	AP	AP ₅₀	AP ₇₅	AP	
EfficientDet-D0 (512)	34.6	53.0	37.1	34.3	3.9M
YOLOv3 [34]	33.0	57.9	34.4	-	-
EfficientDet-D1 (640)	40.5	59.1	43.7	40.2	6.6M
RetinaNet-R50 (640) [24]	39.2	58.0	42.3	39.2	34M
RetinaNet-R101 (640)[24]	39.9	58.5	43.0	39.8	53M

Sono stati presi in considerazione due modelli di Object Recognition:

EfficientDet-D0 e EfficientDet-D1



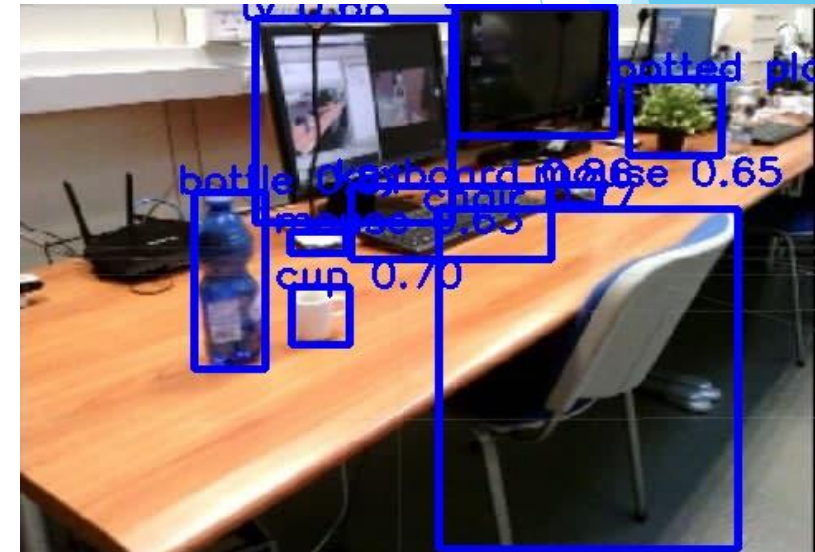


MODELLO (2)

È stato scelto il modello *EfficientDet-D1-coco17* con 32 TPUs (Tensor Processing Unit) e con le seguenti caratteristiche più significative:

Model Name	Speed (ms)	COCO mAP	Input size	Outputs
EfficientDet-D1	54	38.4	640x640	Boxes

- Sebbene *Efficientdet-D0* avesse una velocità di esecuzione più veloce di 15ms, *Efficientdet-D1* presenta un mAP maggiore con una dimensione di ingresso più grande.
- La rete segue il paradigma one-stage detector e usa come rete di backbone EfficientNet pre-addestrata su Imagenet e BiFPN per l'estrazione delle features.



VIDEO



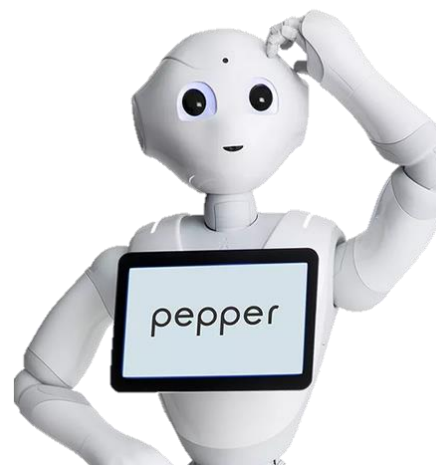
GitHub Project Link: [Cognitive Robotics Project 2020](#)

COGNITIVE ROBOTICS



GRUPPO 18

PEPPER SOCIALE



GRAZIE PER L'ATTENZIONE

