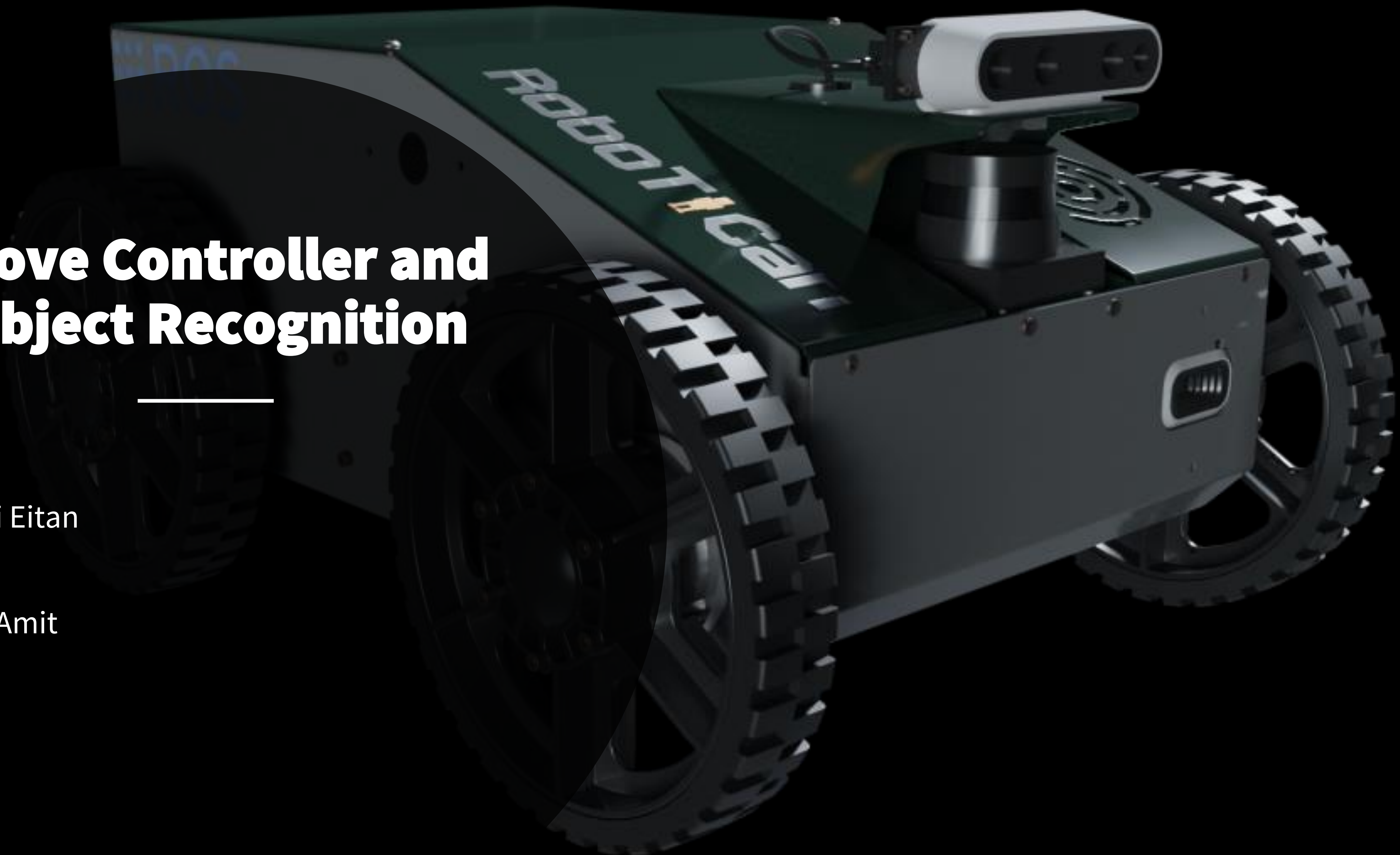


Glove Controller and Object Recognition

- Omri Eitan
- Guy Amit

Goals

- Control the Lizi robot movements, using a remote device via radio signals.
- Detect and recognize objects with Lizi’s camera, using YOLO.
- Receive and display results on remote computer, via wifi.

Motivation

- Our implementation of the glove controller is capable of sending complex commands to robots and other devices, with up to 6 parameters, using an IMU sensor – position x,y,z and orientation x,y,z. In a way that's as simple as moving your hand.

Motivation

- In this project, we use 2 parameters to drive Lizi around. Other uses could be:

- Helping disabled people be more independent.
- Control multiple devices at home or at your workplace remotely with ease.
- Help people work in hazardous environments.

- And that’s with only one of your hands!

Motivation

- We have also implemented the Object Recognition ability of YOLOv3 in ROS environment.
- So our project’s functionality extends even further:

- Recognize and defuse specific bombs.
- Help detecting flaws and hazards in a construction site.
- Find and arrange products in storage warehouses.

How it works

- When the glove is connected to a battery, it start running the Arduino code.
- The Arduino code receive data from the IMU sensor, build a message relevant to our algorithm, and send it using the XBee radio module.
- it’s pair Xbee, connected to the robot’s computer, receive it and use it to command wheels movement.

How it works

- While the robot moves around, the ROS algorithm in it’s computer receive the video stream from the front camera.
- The algorithm then process the image using YOLOv3, and publish the output image as a ROS topic via wifi, using TCP/IP.
- Finally, a remote computer receive the processed image by reading the ROS topic, and display it on the screen.

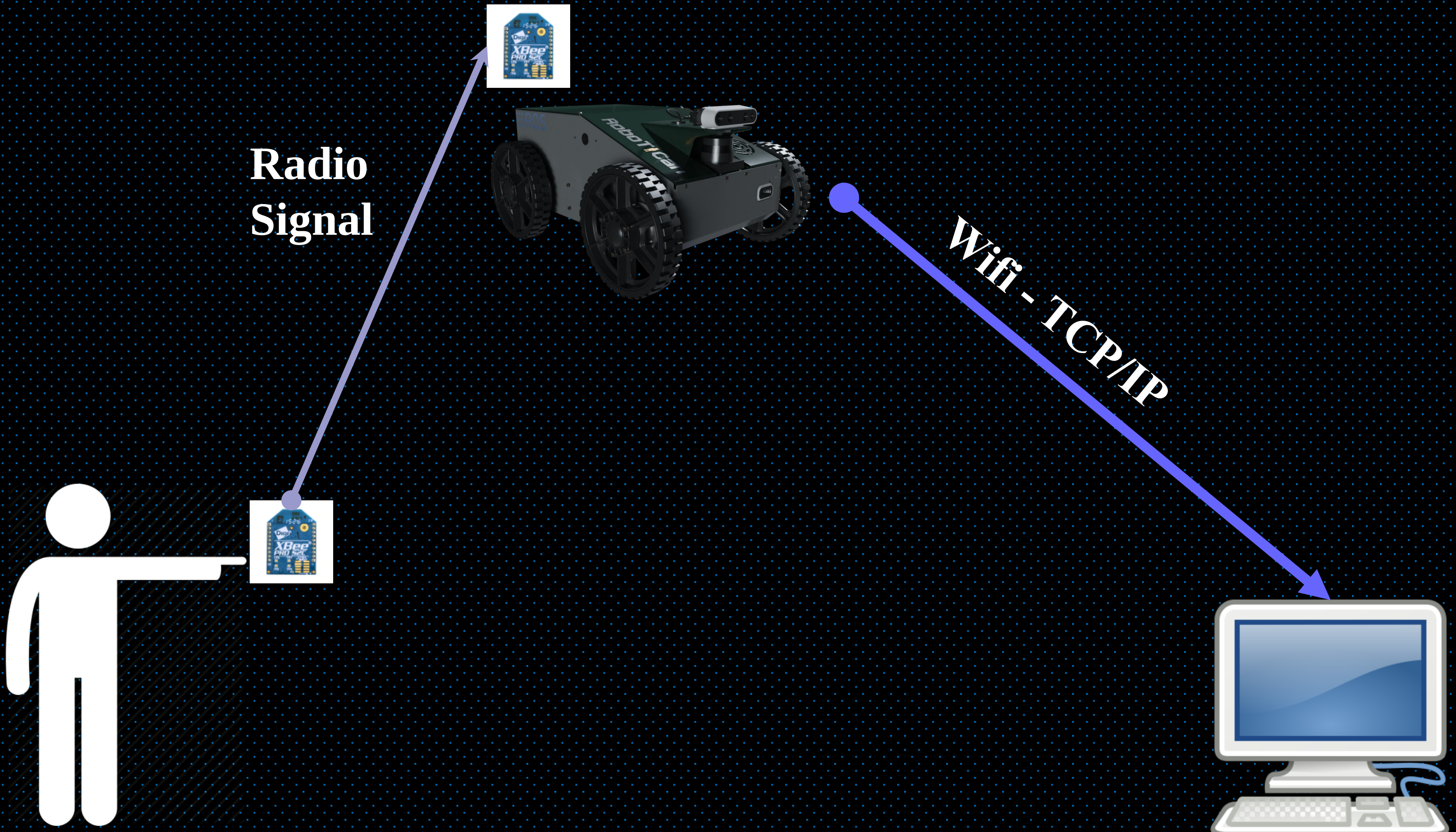
Arduino

- Arduino is an open-source electronics platform based on easy-to-use hardware and software.
- Arduino boards are programed using their own programming language, which has many useful libraries for high-level, easy to learn coding.
- In this project, we programed the glove’s RIC board with Arduino Teensy.

Arduino

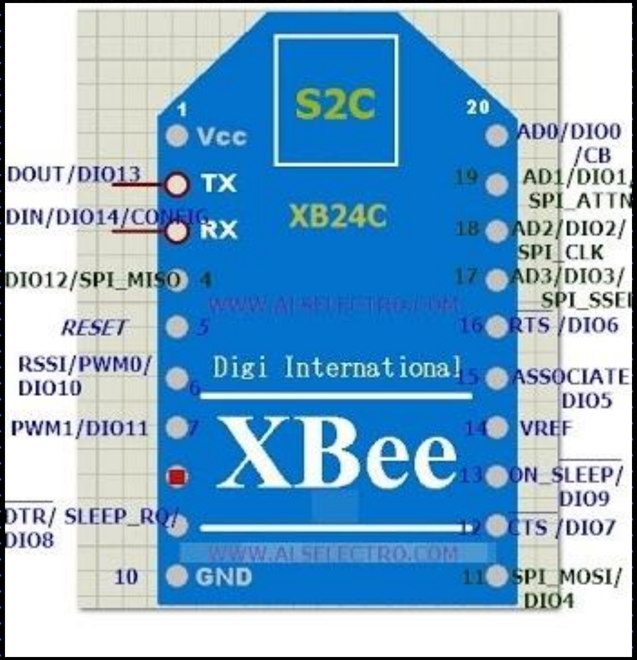
- Our Arduino program main functions are:
-
- Reading the data from the on-board IMU sensor.
- Establish communication between the XBee modules, via serial communication.
- Create custom messeges, containing the processed data, and send them to the Lizi computer.

communication



communication

- We use the rosserial package
- to send a custom message made of
- two float numbers that represent
- roll and pitch orientation of the IMU
- sensor, on the glove.
- The message is sent in serial
- communication to the robot’s
- computer, using the Xbee s2c
- modules.



communication

- After the image is processed on Lizi’s computer, a new ROS topic is created - ‘yolo/output’ and published.
- The topic publishes our custom message, defined by ‘lizi_imu.msg’ in our ROS package.
- Messages are sent in TCP/IP protocol via wifi connection to the remote computer.
- The remote computer subscribe to ‘yolo/output’ topic.
- Every time an image is received, we display it on the screen to show a video stream of the processed image.

ROS – Robot Operating System

- ROS is an open source, meta-operating system.
- It provide frameworks that allow us high level robot programming.
- In this project, we use the ROS communication infrastructure, to publish and subscribe different types of message streams such as:
- - our custom message from the glove.
- - image from the camera, and then to the remote computer.
- - velocity commands to the robot’s wheels

ROS – Packages

- ROS software is organized in packages.
- Those might contain nodes, datasets, configuration files, etc.
- The goal of these packages is to provide useful functionality in an easy-to-consume manner so that software can be easily reused.
- In this project, we built a package to handle both glove and object detection algorithms.
- Our package also interact and uses other packages to achieve our goals.

ROS – Nodes

- ROS nodes are the executables, they are pieces of code where all the algorithm is written.
- In our project, we built two nodes:
- glove_node – subscribe to ‘imu_glove’ topic, and use the data to build and publish velocity messages to the wheels.
- yolo_node – subscribe to the image stream from the camera, process it using YOLOv3 to recognize objects, and publish the output to ‘yolo/output’ topic.

ROS – roslaunch

- Roslaunch is a ROS package, that contains the roslaunch tools, which reads the .launch XML file format.
- A launch file is an XML that describe the nodes that should be run, parameters that should be set, and other attributes of launching a collection of ROS nodes.
- In this project, we made two launch files:
- - one for the glove interface
- - another for the YOLO implementation

Intro to Neural Networks and Yolo

- Intro to neural nets and the brain
- What is Darknet ?
- What is Yolo ?
- what is tiny yolo ?
- OpenCV 3.4 neural nets backend integration with Ros

Intro to Neural nets and the Brain

Motivation and explanation

Neural networks motivation

- Say someone asks you for a lot of money, and you don’t know wheatear you should lawn him money or not. A reasonable thing to do is to try to estimate whether he would return you the money or not. Such estimates are done using looking at previous relevant data regarding people with similar state. i.e. given a set of features regarding the person that is asking for a lawn and the information weather they return it or not. we could deduct some useful information about new people asking for a lawn with similar state.
- Algorithm that do such thinks are called supervised learning algorithms, most of them are based on robust statistics and data analysis. Those algorithms creates what we call classifiers, given datasets containing tuples of features and their correct labels. Those classifiers can predict labels for unseen data i.e. new sets of features.

Neural networks motivation

- Formal definition: Let $\mathbb{D}$ be a distribution over $\mathbb{X} \times \mathbb{Y}$ were $\mathbb{X} \subseteq \mathbb{R}^d\ or\ X \in \mathbb{R}^{n \times m}$ and $\mathbb{Y}$ is a finite set of labels. A classifier C is a function $C: \mathbb{X} \rightarrow \mathbb{Y}$ that given an $x \in \mathbb{X}$ can link x to his correct label in $\mathbb{Y}$.
- Some learning algorithms like neural networks have the ability to learn and predict from vector shaped labels.
- Another name for a set of such functions is hypothesis classes. There are a lot of hypothesis class such as K-nn, Decisions tree and SVMs. One very popular hypothesis class that has shown outstanding results in the past five years is Neural networks.
- Neural networks try to mimic the brain decision making process. Which seems like a reasonable thing to do considering that the human brain is one of the most powerful on earth.

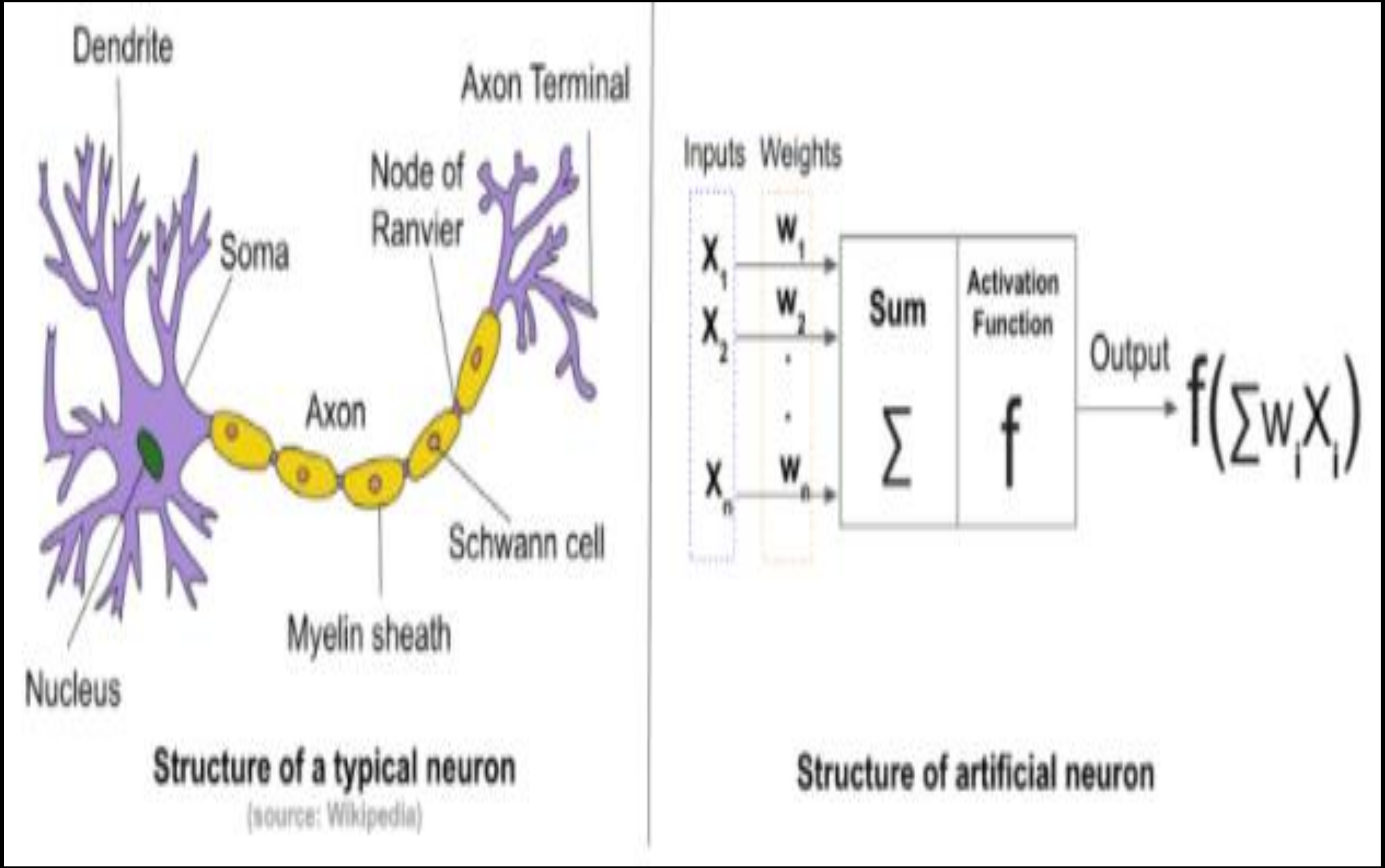
Neural Nets and the Brain

- In the brain, a typical neuron collect signals from others through a fine structures called dendrites. The neuron sends out spikes of electrical activity through the axon (the output and conducting structure) which can split into thousands of branches. At the end of each branch, a synapse converts the activity from the axon into electrical effects that inhibit or excite activity on the contacted (target) neuron. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity (an action potential) down its axon.
- Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

These general properties of neurons can be abstracted in order to study the dynamical behavior of large ensembles of neuronal cells. Thus there have been many interesting attempts to mimic the brains learning processes by creating networks of artificial neurons.
This approach consist in deducing the essential features of neurons and their interconnections and then, programming a computer to simulate these features.

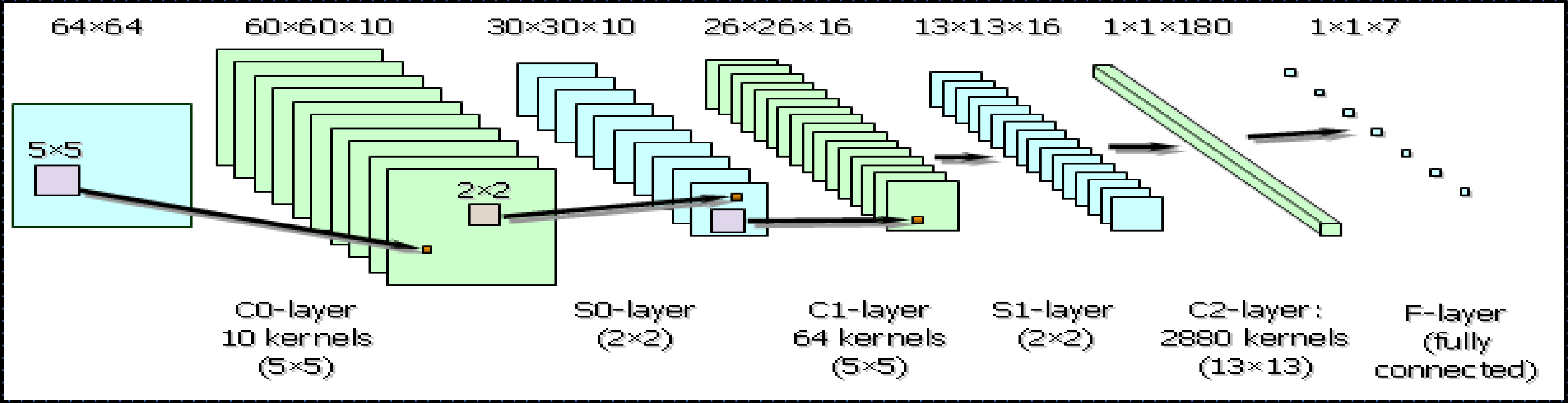
Artificial neural networks are typically composed of inter connected units which serve as model neurons.
The synapse is modeled by a modifiable weight associated with each particular connection. Most artificial networks do not reflect the detailed geometry of the dendrites and axons, and they express the electrical output of a neuron as a single number that represents the rate of firing.

Visual representation of artificial neural



- So far we’ve discussed the similarities to the brain, but through time researches found ways to improve those artificial networks for certain usages.
- One of the most interesting features of the brain is the ability to let us do very complex analysis of images coming from the retina and deduce useful information from them.
- As mentioned before the brain usage weighted sums of its inputs to calculate new signals for other cells, and the artificial neural nets indeed preform weighted sums over the inputs, but as a column of inputs not as a matrix-or an image.
- vision scientists thought it will be a good idea to use convolutions and other kernel based transforms on the input image coming into the neural net instead of reshaping it into a column and then feeding it to the net.

Convolutions based neural networks


64×64
60×60×10
30×30×10
26×26×16
13×13×16
1×1×180
1×1×7
5×5
2×2
C0-layer 10 kernels (5×5)
S0-layer (2×2)
C1-layer 64 kernels (5×5)
S1-layer (2×2)
C2-layer: 2880 kernels (13×13)
F-layer (fully connected)

- This new architecture makes it possible for the networks to learn complex operators like Sobel filter or the Laplacian operator that are both used to detect edges in pictures. And improved significantly the image classification ability of neural nets
- The picture above depicts the architecture of the famous ConvNet named LeNet that is used for various image classification tasks.

What is Darknet?

What is Darknet ?

- Darknet is an open source neural network framework written in C and CUDA. It is fast ,easy to install, and supports CPU and GPU computation. Read more about Darknet: GitHub
- One of the things that Darknet provide are pre-trained weights for several, very deep convolutional neural networks. E.g. Darknet already trained models for various tasks using very powerful GPUs and a very large amounts of data.
- Training such weights on a home computer will take days, or even weeks, and that’s why using a pre-traind wights makes sense when wanting to deploy fast, N.N. based projects or when you are in of short of data for your problem. See transfer learning

YoloV3 –You Only Look Once algorithm

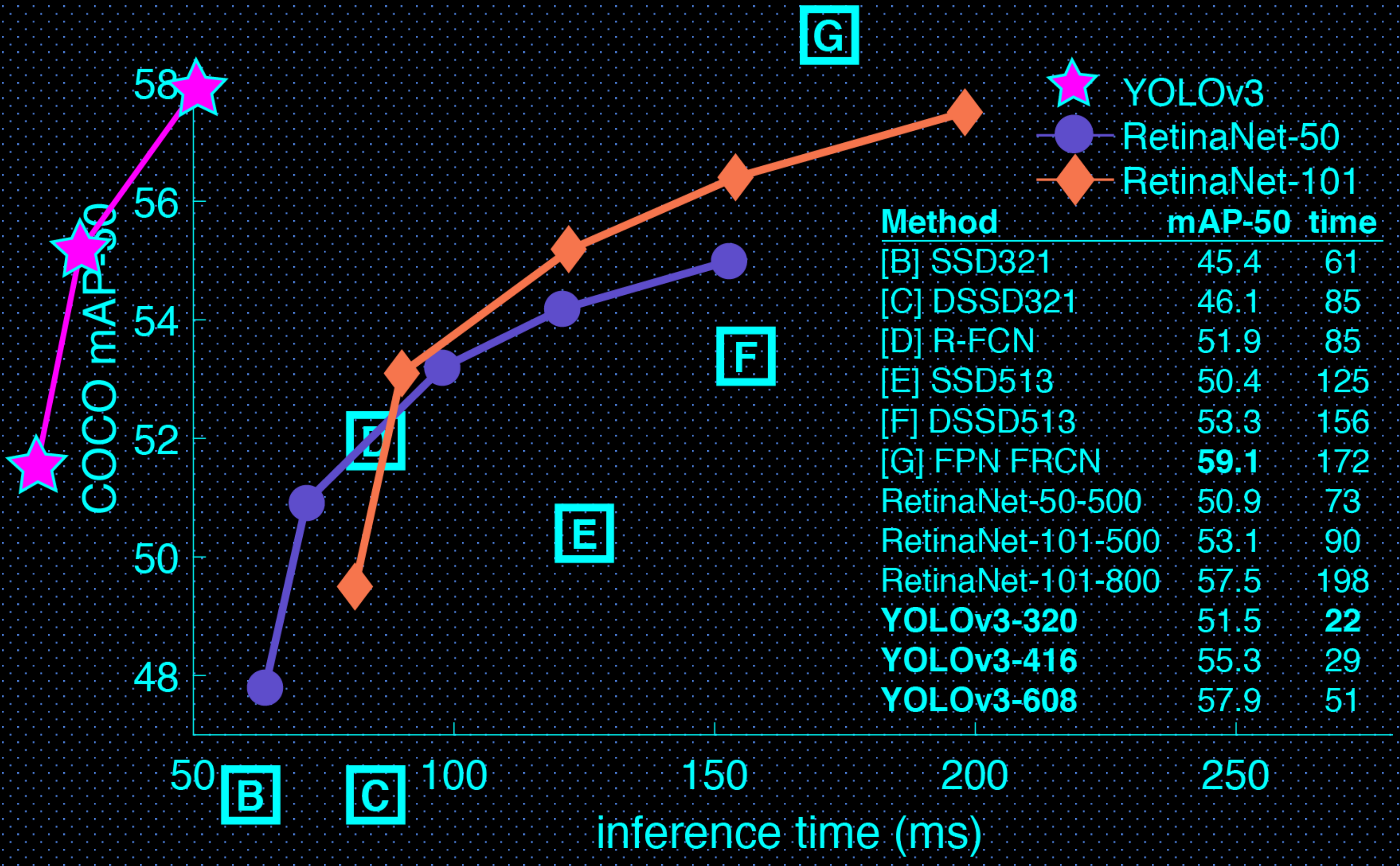
Yolo algorithm

- We have talked about neural networks, in general, but didn't talk about what we can achieve when using them.
- Neural networks are good hypothesis class for a lot of problems such as image recognition, classification, making smart business decisions (invest vs don't invest or lawn vs not lawn), but in our project we will focus on object recognition + object localization, i.e. we want not only to tech the computer the ability to tell us which objects appear in an image, we also want the computer or in our case the Lizi-robot to point to **where** those objects are.

- Over the years several object recognition + localization algorithms were developed, a lot of them uses neural networks, but they were very slow and were not fit to Realtime tasks. Specially on home computers. they all required a large computational power to operate smoothly.
- When the Yolo pepper came out, it was a game changer. Yolo is a state of the art neural net based object recognition +localization algorithm that can run smoothly on most of home computers with reasonable results.
- Yolo stands for You Only Look Once, the algorithm in contrast to other object recognition algorithms, need to feed the image only once through the neural net and then preform post analysis of the results. Making it much faster than other algorithms.

Speed chart

- mAP –mean average precision object detection



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

The Yolo algorithm pseudo code:

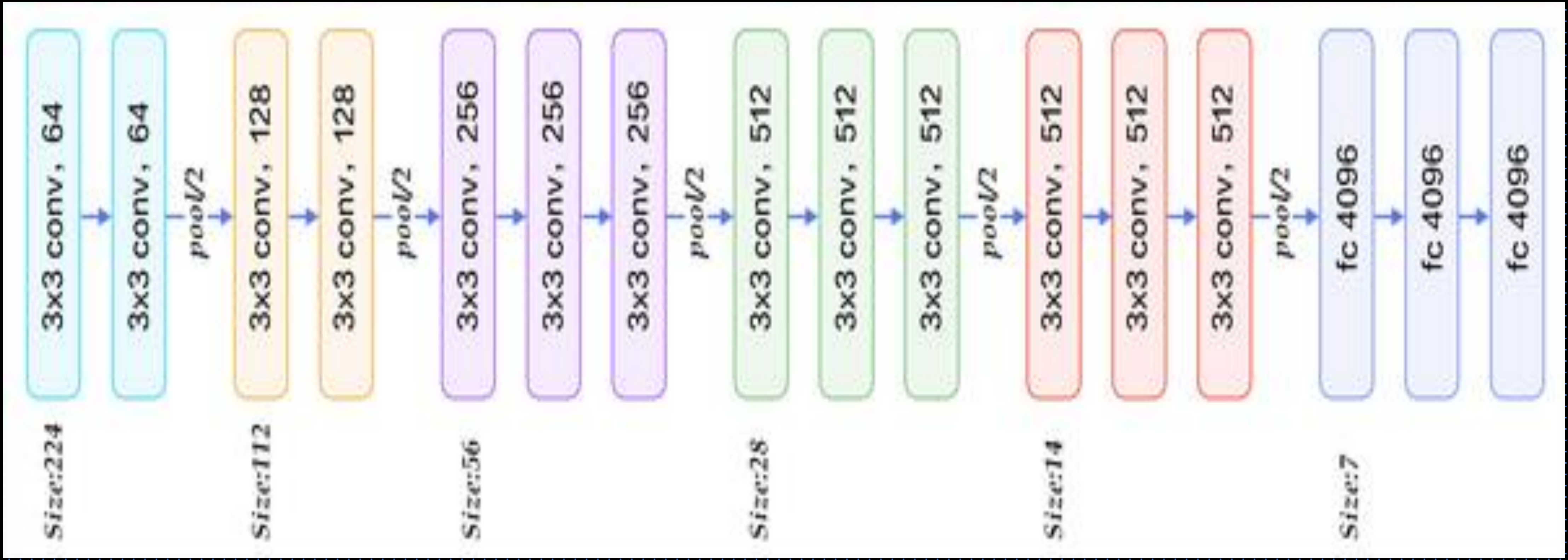
1. Divide image into a $S \times S$ grid
2. Feed each sub image into a convolutional Neural network that predicts bounding boxes and labels for each of the classes
3. Collect resulting predications and use the confidence paramater that was supplied to the model to threshold the overlapping bounding boxes using the non-max suppression algorithm
4. Output a list of bounding boxes and labels

Tiny Yolo and VGG16

- Though the Yolo algorithm is the fastest object recognition out there, it still run slow on our robot due to the fact that we do not have a Nvidia graphical processing unit. And most machine learning applications require one.
- Luckily we have found a solution to the problem: Tiny yolo
- Tiny Yolo is a smaller version of yolo that uses the weights of a shallower neural network called VGG16. this neural net is less accurate than the standard neural networks that yolo works with but it is mush faster and enable us to process video in real time.

Tiny Yolo and VGG16

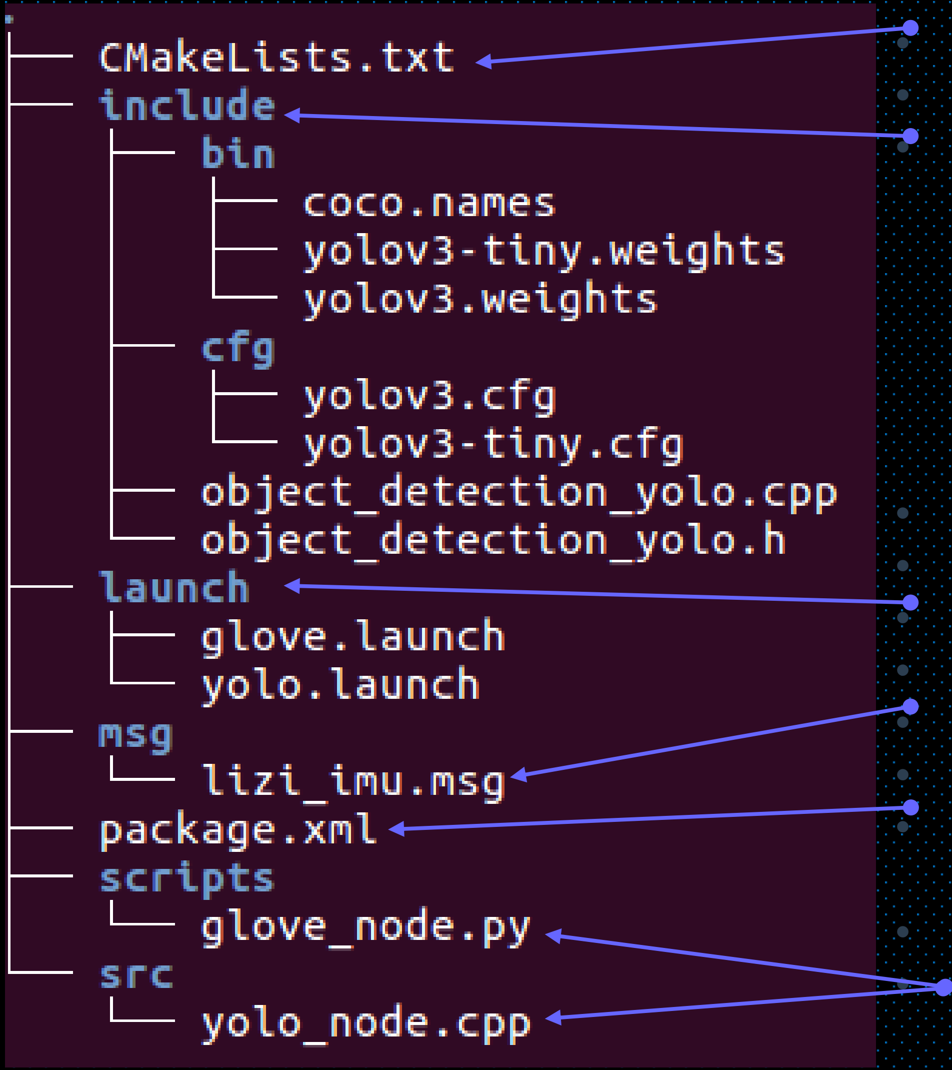
The architecture of the VGG16 neural network



OpenCv 3.4 neural nets integration with ROS

- As mentioned before, running the regular yolo version is pretty slow on the robot. So we have searched for the fastest way to run the data through the VGG16 net and get results
- Unfortunately the Darknet interface is closed and only optimized for heavy GPU machines. And that’s why we have decided to use the Darknet weights of the VGG16 model for Yolo and run them with OpenCV3.4.4 neural networks back end which is highly optimized for intel CPU’s.

Our package - yolo_lizi



```
.
├── CMakeLists.txt
├── include
│   ├── bin
│   │   ├── coco.names
│   │   ├── yolov3-tiny.weights
│   │   └── yolov3.weights
│   ├── cfg
│   │   ├── yolov3.cfg
│   │   └── yolov3-tiny.cfg
│   ├── object_detection_yolo.cpp
│   └── object_detection_yolo.h
├── launch
│   ├── glove.launch
│   └── yolo.launch
├── msg
│   └── lizi_imu.msg
├── package.xml
├── scripts
│   └── glove_node.py
└── src
    └── yolo_node.cpp
```

Compilation instructions

Include non-ROS pieces of code we need for the YOLOv3 algorithm

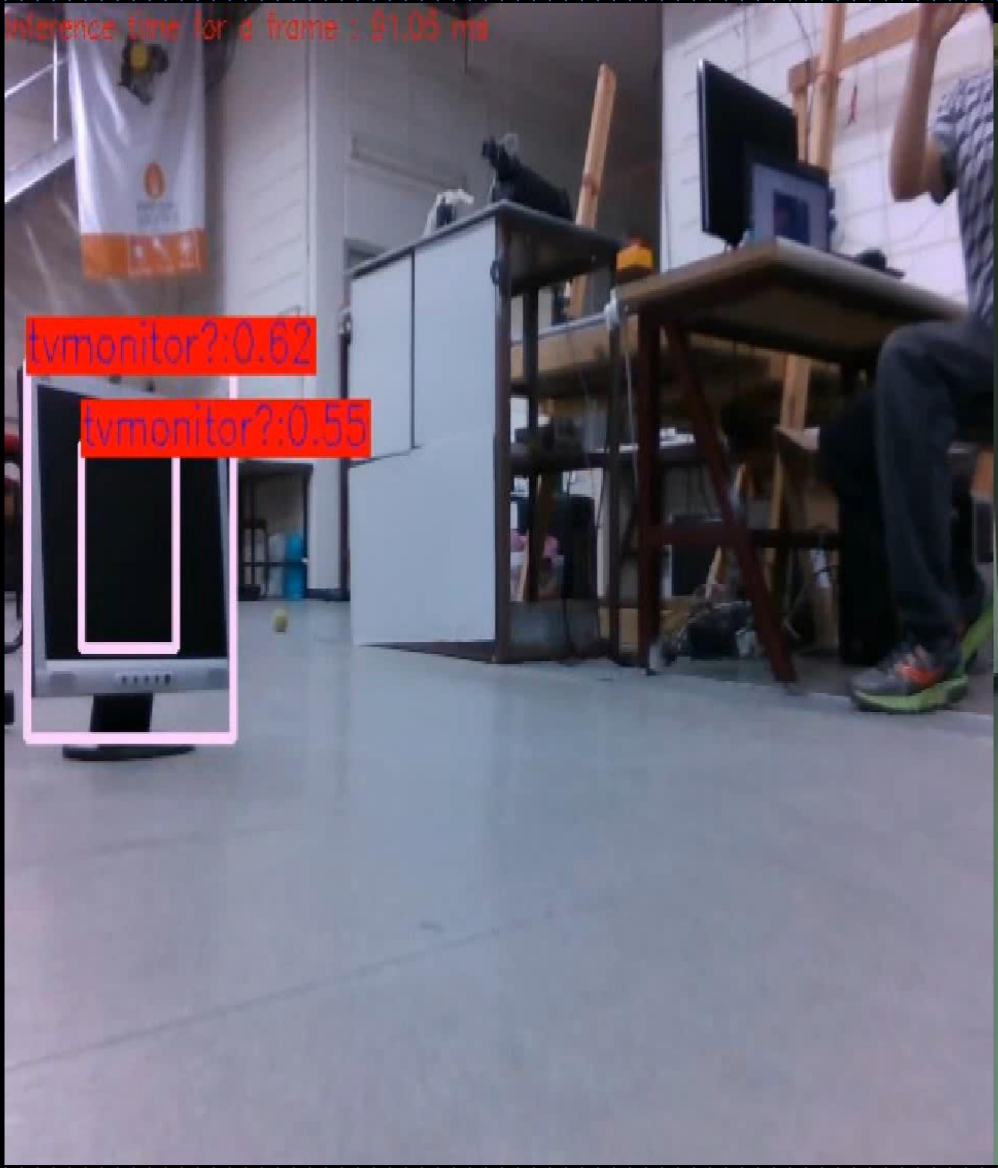
Launch files

Our custom message

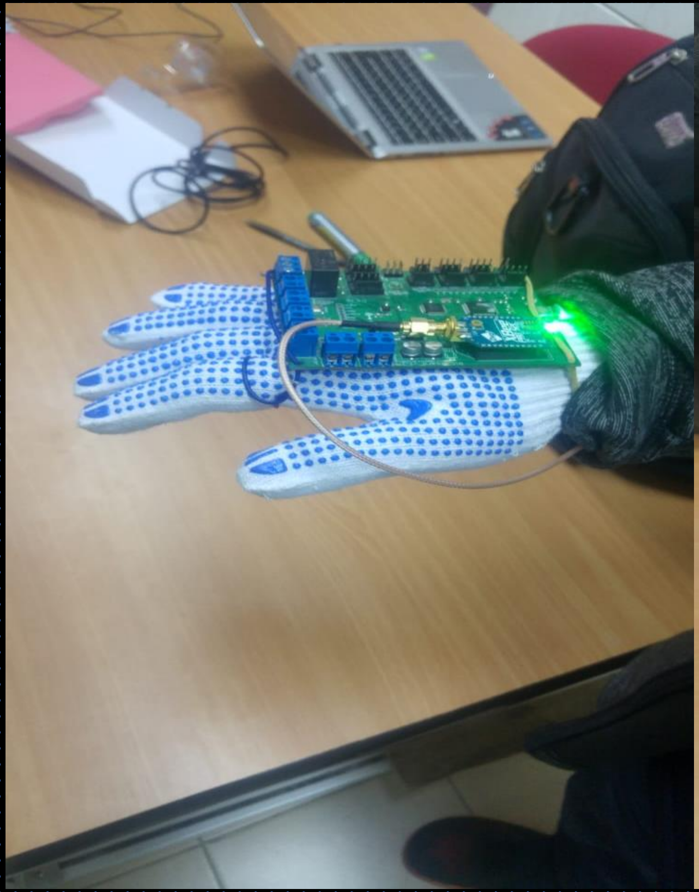
Package manifest - defines properties such as the package name, authors, and dependencies.

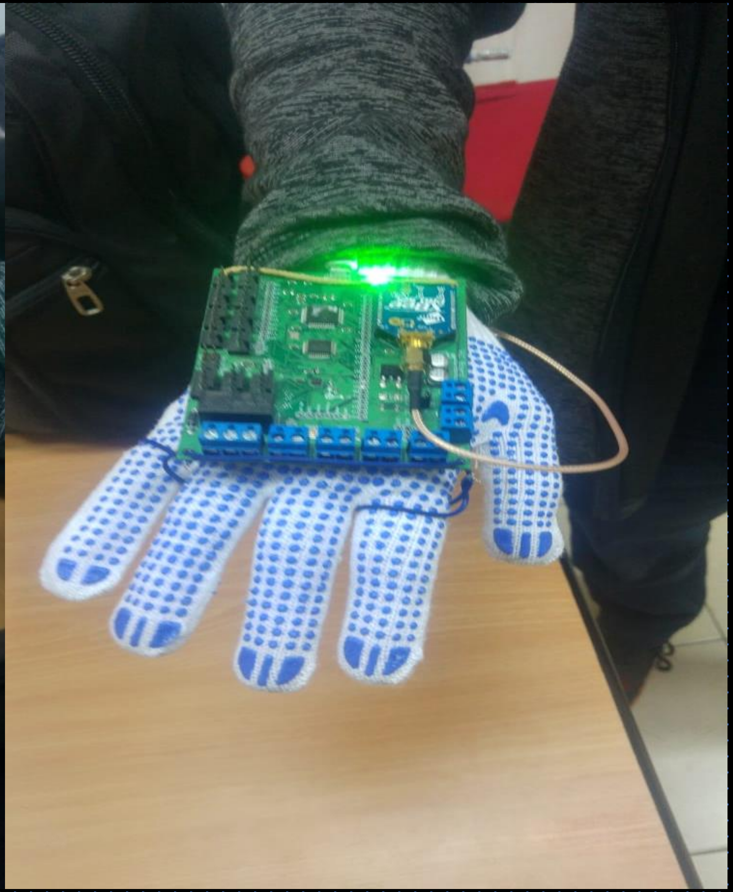
The nodes

DEMO


Inference time for a frame : 91.05 ms
tvmonitor?:0.62
tvmonitor?:0.55

The glove

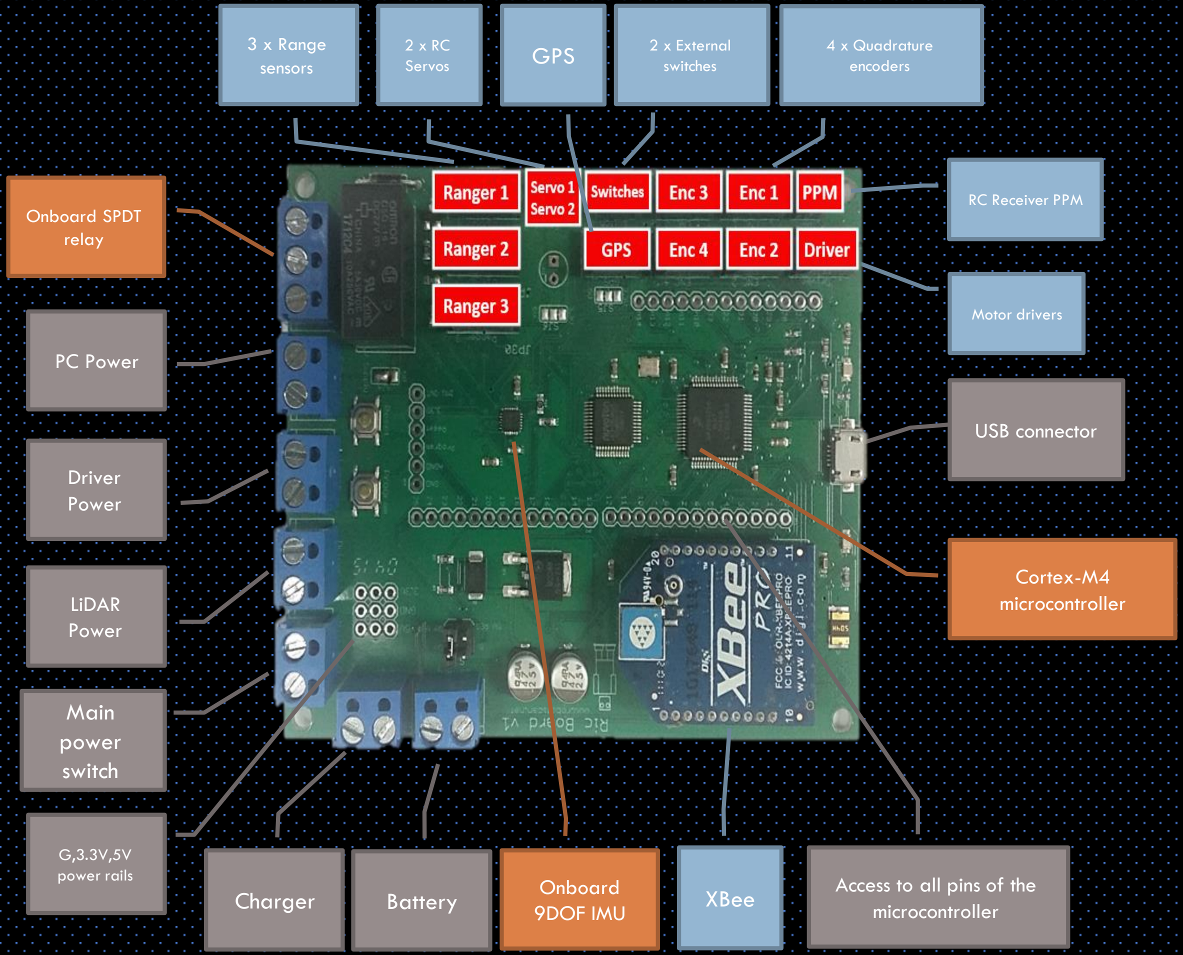




Hardware – RIC board by Robotican Ltd.

Main Robot Controller for DIY Robot

multi-purpose robot controller which serve as the interface between the robot's motors, sensors, and a ROS based computer.



Hardware – RIC board

- We used the RIC board for our glove controller
- We connected a small battery via the USB connector
- Used the readings from the on-board IMU sensor for our messages.
- And connected an XBee module as seen in the photo.

Lizi Robot

Designed to fit academic research and budget!

The **Lizi Robot** is a smart mobile robotics system. It features a skid steering rover, a complete sensors suite for indoor and outdoor navigation, and a native interface to ROS. The ROS package contains all the drivers and programming examples.

The Lizi robot is designed to work alone or as part of a group of robots.

ROS.org

The **Lizi** robot features:

- Dimensions: 38 L x 28 W x 22 H cm (15" x 11" x 8.7")
- Designed to work both **indoor** and slightly **outdoor**
- Fully **ROS** compatible http://wiki.ros.org/lizi_robot
- Maximum continuous operation time of **2 hours**
- Designed to work as a **group of Lizi Robots**
- Sensors: **IMU**, **Compass**, **GPS**, **camera**, **RGB-D Camera mounted on Pan-Tilt unit, Ultrasonic range sensors, Laser scanner.**
- Contains **microphone and speaker** for vocal interaction with humans or for surveillance
- **Customization** and **modifications** available

Lizi Robot Specifications

- Dimensions: 38 L x 28 W x 22 H cm
- Weight: 9.1 Kg (with battery).
- 4 solid rubber wheels, differential drive.
- Wheels diameter: 14 cm.
- Powerful, 4 x 50W motors.
- High resolution encoders (4288 counts per revolution) for closed loop control.
- Maximum speed: 3.6 Km/hr.
- Battery: 12V SLA 9Ah (Charging socket available on the robot, Charger included).
- MTBC: 2 Hours
- Payload: 2 kg

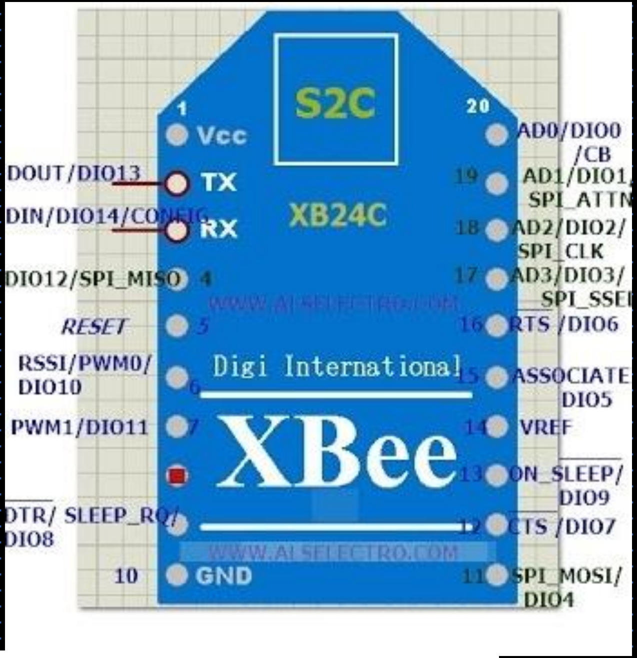
- Onboard Intel i5 **computer**.
- 120GB Solid State Drive (SSD)
- Built –in 802.11 ac Dual Band **WiFi** + **Bluetooth** 4.0
- Sensors: **GPS**, 9 DOF **IMU** (3 Gyros, 3 Accelerometers, 3 Magnetometers), 3 **URF** - Ultrasonic Range Finders mounted on the sides and back of the robot
- **Laser** Rangefinder
- **RGB-D** sensor mounted on pan-tilt system (Optional)
- Front **camera**
- Contains **microphone and speaker**
- **Full computer panel** on the back allows connection of USB devices, Joystick, Keyboard, Mouse, Monitor

Hardware – the Lizi robot

- For this project We used the Lizi robot, made by Robotican Ltd.
- features we used:
- - image from front camera
- - wheels movement
- - basic ROS packages
- Everything else regarding, described in this document is implemented / assembled by us.

Hardware - XBee

- We used two XBee s2c modules
- to send and receive the radio signals.
- One connected directly to the RIC
- board on the glove
- The other connected to Lizi’s
- computer via USB.
- To connect XBee module via USB,
- we used XBee Explorer.



S2C
XB24C
Digi International
XBee
1
Vcc
DOUT/DIO13
TX
DIN/DIO14/CONFIG
RX
DIO12/SPI_MISO
4
RESET
5
RSSI/PWM0/DIO10
6
PWM1/DIO11
7
DTR/SLEEP_RQ/DIO8
10
GND
20
AD0/DIO0/CB
19
AD1/DIO1/SPI_ATTN
18
AD2/DIO2/SPI_CLK
17
AD3/DIO3/SPI_SSEL
16
RTS/DIO6
15
ASSOCIATE/DIO5
14
VREF
13
ON_SLEEP/DIO9
12
CTS/DIO7
11
SPI_MOSI/DIO4