POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Thesis

Semantic Simultaneous Localisation based on deep-learning algorithm, Mapping and Tracking

Supervisors
prof. Berger Cyrille
prof. Caputo Barbara

Alberto Giacomini

April 2020

Semantic Simultaneous Localization based on deep-learning algorithm, Mapping and Tracking
Master thesis. Politecnico di Torino, Turin.

April 2020.

Acknowledgements

I would like to thank my two supervisors Cyrille and Barbara, for their guidance through each stage of the process and for the patience towards me. With them, all the guys from the lab and from the department in Linköping because they treat me like one of them and they gave me all the support I needed. All the people I met while I was far from Italy because each of them, with its stories and the moments shared together, has left something positive in me and they allowed me to have a second family far from home.

Ora vorrei ringraziare chi più di tutti ha sempre creduto in me, i miei genitori e la mia famiglia. Grazie perchè ci siete sempre stati e grazie perchè mi avete sempre spronato ad arrivare dove sono ora, chi in un modo e chi in un altro, senza avermi mai fatto pesare nulla. Ho sempre potuto contare su di voi come esempi e modelli da seguire. Grazie anche ai miei amici e a coloro che mi hanno accompagnato in questi anni e con cui ho vissuto momenti indimenticabili, supportandomi e sopportandomi per qualsiasi cosa. Grazie ai miei coinquilini e ai nuovi amici che ho conosciuto a Torino perchè con voi non ho sentito la distanza da casa. Grazie a Giulia, perchè con lei ho condiviso la maggior parte del mio percorso finale, attraversando parecchi momenti facili e difficili raggiungendo diversi obiettivi universitari e non, ma sempre con il suo supporto al mio fianco.

Grazie di cuore a tutti, perchè ognuno di voi ha lasciato un tassello importante di se, in me. Contribuendo così a formare la persona che sono ora.

Abstract

During the last few years, the area of greatest development has been that of Artificial Intelligence (AI), given the fact that it is applicable in the vast majority of situations that also involve those of daily life. The field in which it is applied by us is the robotic one where, a fundamental point, is the perception of the environment by the robot. The problem that has been analyzed is the Simultaneous Localisation And Mapping (SLAM) which has been addressed through the integration of deep-learning algorithms such as object detection. In this thesis is analyzed and compared the state of the art of those algorithms and is given a solution as robust as possible to the problem stated above.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A key challenge in mobile robotics is the perception of the environment and the availability of an accurate map allows for the design of systems that can operate in complex environments only based on their on-board sensors and without relying on external reference system (i.e. GPS). A lot of work has been done on a subject called "Simultaneous Localisation And Mapping" (SLAM). This problem asks if it is possible to construct or update a map of an unknown location in an unknown environment while simultaneously keeping track its location within this map.
In SLAM both the trajectory of the platform and the location of all landmarks are estimated on-line without the need for any a priori knowledge of location.
SLAM has been formulated and solved as a theoretical problem in several different forms. Those techniques are quite well established in the case of a static environment (i.e. no moving obstacles) and predefined landmarks. The biggest problem is for the robot to decide what is static and what is moving. In our project also what is a landmark and what is not. A possible approach is to consider that everything is moving, and if after many iterations, an object is not moving changing it to a static object. But this lead to poor estimation of the robot position and lower the quality of the map. What we would like is to investigate detecting if the object is likely to move based on observation.
This observation is based on a set of cameras mounted on the machine. Considering that, a possible approach would be to use a real-time deep-learning

based image classifier to identify and classify objects in the environment. After finding the class we can mark them as "moving object" or "static object", as “landmark” and “not landmark” (i.e. a human is a moving object, and a tree is static and possible landmark object).
Then, using a graph-slam approach, after process data, it would be possible to build the map, localise the robot and track the moving objects.

1.2 Aim

This project aims to integrate, on a ground robot, an object detection algorithm with a graph-SLAM approach in order to improve the map building and the perception of the environment from the non-humane unity by the combination of object tracking and object detection.

1.3 Research questions

The research questions we are aim to answer are:

1. Which between YOLO and RetinaNet is the best real-time object detection CNN?

2. How is, the distance measurement through the camera affected, compared to other instruments?

3. How it will affect, the implementation of the tracker, the full system. How it can be correlated to the map building in the graph-SLAM approach?

4. How does the object detection approach will increase the accuracy level of the map building in the graph-SLAM approach?

Chapter 2

Basic concepts of Artificial Intelligence and Machine Learning

The purpose of this chapter is to introduce basic helpful concepts to understand the context of the thesis.

2.1 Artificial Intelligence

"The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." [1]

Artificial Intelligence (AI), theory and algorithms that enable computers to mimic human intelligence. Such as the ability to reason, discover meaning, generalize, or learn from past experience, AI is one of the newest fields in science and engineering, coined after the World War II in 1956. [2]

After its foundation it was applied in different subfields from the proving of mathematical theorems to a car driving through healthcare, video games and military.

Different approches can be follow. Statistical methods, computational intelligence, and traditional symbolic. Many tools are used in AI, including versions of search and mathematical optimization, artificial neural networks, and methods based on statistics, probability and economics. The AI field

draws upon computer science, information engineering, mathematics, psychology, linguistics, philosophy, and many other fields.

After its foundation[1] several philosophical discussion were raised on the ethics of using AI to simulate human thought by creating artificial beings. These have been explored by myth, fiction and philosophy. Some people also consider AI dangerous to humanity if it progresses without control.

2.2 Machine Learning

We are entering the era of big data. For example, there are about 1 trillion web pages[3]; one hour of video is uploaded to YouTube every second, amounting to 10 years of content every day 2; the genomes of 1000s of people, each of which has a length of 3.8×10^9 base pairs, have been sequenced by various labs; Walmart handles more than 1M transactions per hour and has databases containing more than 2.5 petabytes (2.5×10^{15}) of information (Cukier 2010) and so on.

This deluge of data calls for automated methods of data analysis, which is what **machine learning** provides.[4]

ML is a subset of AI that includes statistical techniques enabling machine to improve tasks with experience.

It is usually divided into two main types:

1. Predictive or supervised learning: the goal is to learn a mapping from inputs x to outputs y, given a labeled set of input-output pairs $D = \{(X_i, Y_i)\}_{i=1}^N$. Here D is called the "training set", and N is the number of training examples;

2. Descriptive or unsupervised learning approach. Here we are only given inputs, $D = \{x_i\}_{i=1}^N$, and the goal is to find “interesting patterns” in the data, sometimes called knowledge discovery. This is a much less well-defined problem, since we are not told what kinds of patterns we have to look for, and there is no obvious error metric to use (unlike supervised learning, where we can compare our prediction of y for a given x to the observed value).

2.3 Deep Learning

Since the beginning of the AI, the subject solved and faced with problems that are intellectually difficult for human beings, but relatively simple for computers. A real challenge for AI are problems that are easy for people to perform but hard for people to describe formally, problems that we usually solve intuitively, that feel automatic, like recognize faces or spoken words.

One solution to these problems is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts.

The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI **deep learning**. [5]

2.3.1 Neural Networks' basics concept

In order to understand what a CNN is, it is important to know some basic concepts.

Perceptron

The perceptron (2.1) takes for input different binary values x_1, x_2, ..., x_n and produce a single binary output y.

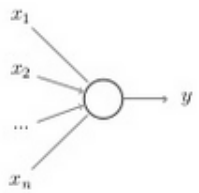


Figure 2.1: Perceptron

Rosenblatt, in 1958 introduced some little rules in order to evaluate the output by the introduction of weights by matching input value x_j, to a weight w_j, a real value that represents the importance of input x_j in the calculation of y.

y can assume values included between 0 and 1 following this formula:

$$y = \begin{cases} 0 & if \sum_i w_j x_j \leq threshold \\ 1 & if \sum_i w_j x_j > threshold \end{cases}$$

One single perceptron cannot represent a robust decision-making model, but a combination of them can take a complex decision based on the easiest ones.

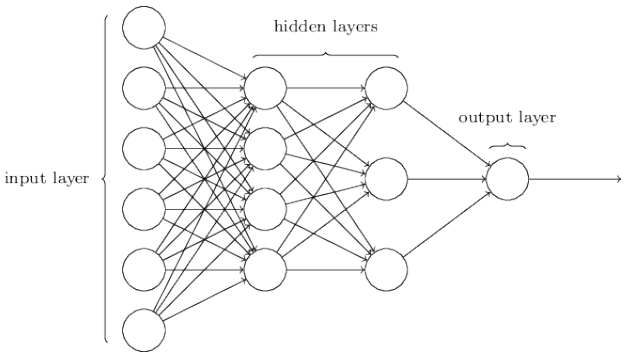


Figure 2.2: Multilayer Perceptron

In the network figure 2.2, more neurons are connected. The first column or *layer* takes easy decisions producing results. These are processed by the second layer that, weight the input, and can takes decision in a higher, abstract and complex level. More convoluted is going to be the decision taken by the third layer and so on.

Following this path, a perceptron's network can solve complex problems.

Learning

The main purpose of the learning of a neural network is to adjust all the weights in the different layers in order to have a proper output from the network.

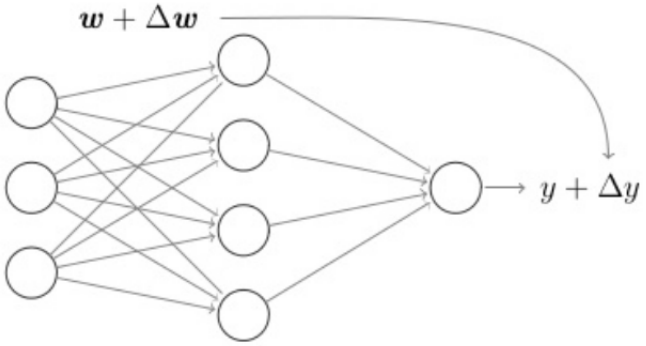


Figure 2.3: Learning process NN

When we are designing a neural network, inputs corresponds to the first layer of neurons whose outputs are the same input values. If small changes in the weight, corresponds to small changes in the output, it would be possible to use this capability to lead the network to the best behavior based on the situation.

For example, if we have a neural network for the classification of images that misclassify a dog image as a "cat" one, a little change on the weights can lead the network to correct classification. This change is what we call learning.

This does not happen with the neural network that contains perceptron; in fact, a small change on the weight can bring the output to change from 0 to 1 and this change can lead to a large change in the behaviour of the network. So even if the classification is correct; it is probable that every other image is changed without control. For this reason, the sigmoid neuron fits better the situation.

Sigmoid Neuron

Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. This fact will allow a network of sigmoid neurons to learn.

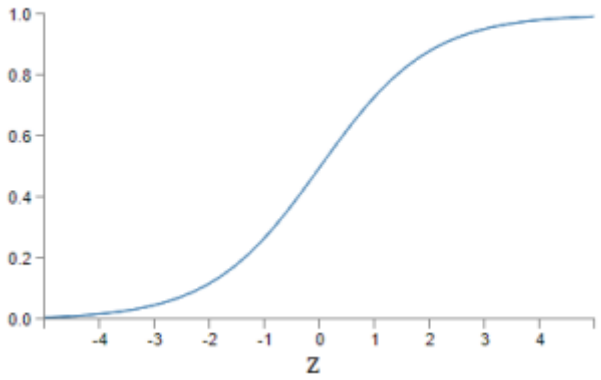


Figure 2.4: Sigmoid Function

The rappresentation of a sigmoid neuron is the same as a normal neuron but the inputs can take any value between 0 and 1. Also like the perceptron, it has weights(w_1, w_2, ...) and overall bias b. But the output is not 0 or 1

but follow this function called *Sigmoid Function*:

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \tag{2.2}$$

Suppose $z = w \cdot x + b$ is a large positive number. Then $e - z \approx 0$ and so $\sigma(z) \approx 1$. In other words, when $z = w \cdot x + b$ is large and positive, the output from the sigmoid neuron is approximately 1, just as it would have been for a perceptron. Suppose on the other hand that $z = w \cdot x + b$ is very negative. Then $e - z \to \infty$, and $\sigma(z) \approx 0$. So when $z = w \cdot x + b$ is very negative, the behaviour of a sigmoid neuron also closely approximates a perceptron. It's only when $w \cdot x + b$ is of modest size that there's much deviation from the perceptron model.

Backpropagation

In the Multi-Layer Perceptron (MLP) the presence of multiple layers of connection imply the use of special weight correction techniques. In the feed-forward network is used the Backpropagation. We identify two phases:

1. Signal propagation [IN -> OUT]
2. Error propagation [OUT -> IN]

Considering a generic network, three types of neurons are identified :

- x_k -> Input layer;
- a_j -> Hidden layer;
- y_i -> Output layer;

and two level of connection:

- W_{jk} -> Connection between x_k e a_j;
- V_{Ij} -> Connection between a_j e y_i;

During the first phase (signal propagation):

- Is presented the μ-th x^μ;

- The hidden layers receive the input

$$h_j^\mu = \sum_k W_{jk} x_k^\mu$$

  and they produce the output:

$$a_j^\mu = g(\sum_k W_{JK} x_k^\mu)$$

- The output layers receive the input

$$h_i^\mu = \sum_j V_{ij} a_j^\mu = \sum_k V_{ij} g(\sum_k W_{jk} x_k^\mu)$$

  and they produce the output(assuming the same activation function):

$$y_i^\mu = g(\sum_j V_{ij} a_j^\mu) = g(\sum_j V_{ij} g(\sum_k W_{jK} x_k^\mu))$$

While in the second phase (error propagation):

- The error function is calculated

$$\begin{aligned} E &= \frac{1}{2} \sum_{\mu i} (d_i^\mu - y_i^\mu)^2 \\ &= \frac{1}{2} \sum_{\mu i} [d_i^\mu - g(\sum_j V_{ij} g(\sum_k W_{jK} x_k^\mu))]^2 \end{aligned}$$

- Is applied the gradient descent for the first connection layer

$$\begin{aligned} \Delta V_{ij} = -\eta \frac{\delta E}{\delta V_{ij}} &= \eta \sum_\mu (d_i^\mu - y_i^\mu) g'(h_i^\mu) a_j^\mu \\ &= \eta \sum_\mu \delta_i^\mu a_j^\mu \end{aligned}$$

  with

$$\delta_i^\mu = (d_i^\mu - y_i^\mu) g'(h_i^\mu)$$

- Is applied the gradient descent for the second connection

$$\begin{aligned} \Delta W_{jk} = -\eta \frac{\delta E}{\delta W_{jk}} &= -\eta \sum_\mu \frac{\delta E}{\delta a_j^\mu} \frac{\delta a_j^\mu}{\delta W_{ij}} \\ &= \eta \sum_{\mu i} (d_i^\mu - y_i^\mu) g'(h_i^\mu) V_{ij} g'(h_j^\mu) x_k^\mu \\ &= \eta \sum_{\mu i} \delta_i^\mu V_{ij} g'(h_j^\mu) x_k^\mu \\ &= \eta \sum_{\mu i} \delta_i^\mu x_k^\mu \end{aligned}$$

with

$$\delta_j^\mu = g'(h_j^\mu) \sum V_{ij} \delta_i^\mu$$

In the case of multiple hidden layers, the error is propagated until the input, updating layer after layer each weight vector.

Rectified Linear Unit - ReLU

In the previous chapter, it can be observe that we need to derivate the activation function. If we derive the sigmoid function sigma (x) we could see that we will obtain, for x that goes to + and - infinite, a gradient that goes to zero. This means that ΔW_{ij} goes to zero and the backpropagation does not converge. This problem is known as the vanishing gradient. To partially solve this problem is introduced the activation function ReLU.

Is defined as the positive part of its argument:

$$f(x) = x^+ = \max(0, x)$$

where x is the input to a neuron.

With this function we can observe that the gradient is not equal to zero for x that goes to infinite.

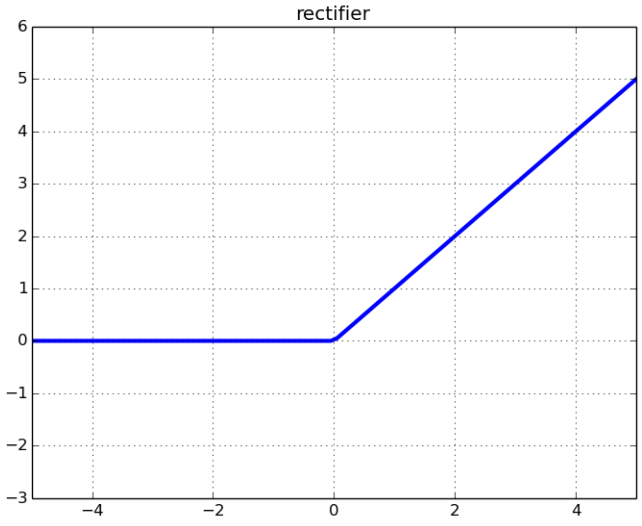
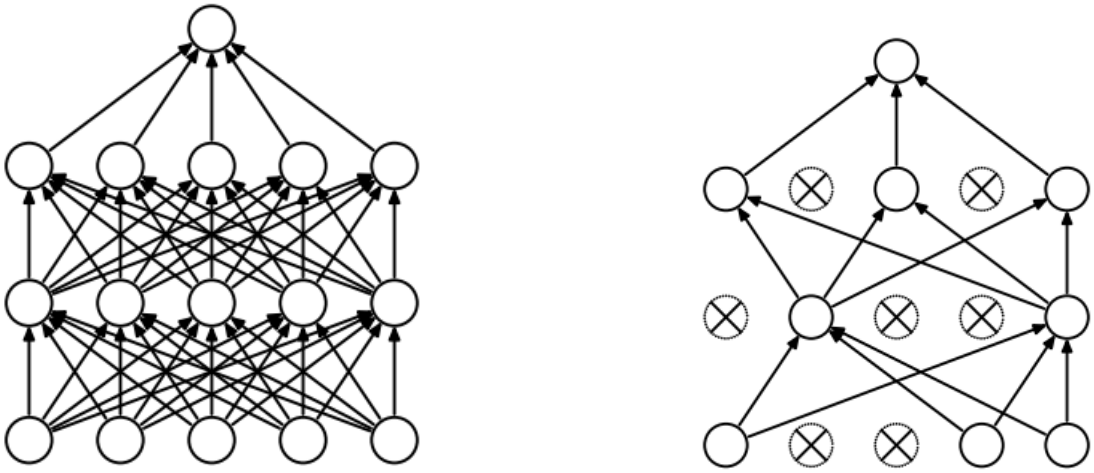


Figure 2.5: ReLU function.

Dropout techniques

Deep neural networks are made by different connections between neurons. A group of neurons is called a layer, some of them are hidden other are visible. The dropout technique aims to face the overfitting problem by dropping out some of these neurons. More easily the dropout randomly ignores some neurons during the training phase. This means that some of them are randomly not considered during the forward or backward propagation.



(a) Standard Neural Network. (b) After applying dropout.

Figure 2.6: Dropout in a Neural Network.

By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections as shown in figure 2.6. The choose of which unit cut, as already said, is random. The probability p is fixed or chosen using a validation set. Typically a good fixed value is 0.5, close to optimal for a wide range of networks and tasks. For what concern input units the optimal probability is more closer to 1 than 0.5. [6]

Data Augmentation

Sometimes we do not have so many data in our dataset. So, it could be difficult to train or have a good performance in our neural network. The data augmentation tries to help with this problem by adding more data on the dataset by means of some mutation to the images, like flipping, rotation, scaling etc.

There are two different types of data augmentation based on different application timing on the pipeline of my CNN:

1. Offline augmentation: is preferred for smaller datasets. Is applied beforehand, increasing the size of the dataset by a factor equal to the number of transformations that are performed;

2. Online augmentation or augmentation on the fly: is performed on a mini-batch, just before feeding the machine learning model. Preferred for larger datasets. Some machine learning frameworks have support for online augmentation, which can be accelerated on the GPU.

The most popular techniques of data augmentation are:

- Scale: the image can be scaled outward or inward. The data augmentation factor is arbitrary;

- Crop: sampling the original image and resize the section to the original image size. The data augmentation factor is arbitrary;

- Flip: can be applied horizontally and vertically. The data augmentation factor goes from 2x to 4x;

- Rotation: only possible problem is that the rotation of the image may not preserve the image size. Usually applied with 90° degrees angles. The data augmentation factor goes from 2x to 4x;

- Translation: the translation is done moving the image along the X or Y direction (or both). This is an excellent method because most of the objects can be located almost anywhere in the image. This leads the CNN to look everywhere. The data augmentation factor is arbitrary.

2.3.2 CNN - Convolutional Neural Network

The Convolutional Neural Network (CNN) is a feed-forward artificial neural network. This use an architecture inspired to the organization of the animal vision, whose neurons, overlapped, are stimulated from a specific region called *receptive field*. The CNNs are inspired by biological processes and are a variation of multi-layer perceptrons developed to use the pre-processing to a minimum. They are used in different fields as image and video detection, natural language recognition and bioinformatics.

Region-based Convolutional Network (R-CNN)

This kind of approach is divided into two phases. The first begins with the region search and then perform the classification on them. In order to deal with the problem of selecting a huge number of regions [7] proposed an alternative to exhaustive search where we use selective search to extract object location.

Figure 2.7: Examples of our selective search application

It generates a sub-segmentation of the image to create many small regions. Using a greedy algorithm, merges them with a hierarchical grouping, according to a variety of colour spaces and similarity metrics, in a larger one. It uses the generated regions to produce the final output region proposals.

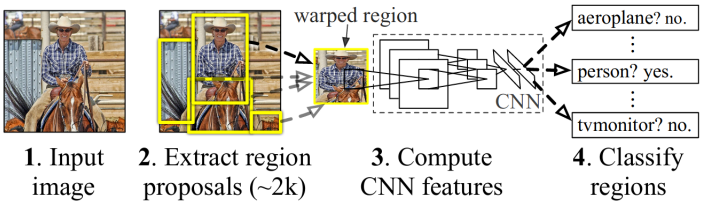


Figure 2.8: Object detection system overview

A combination of selective search methods to detect region proposals and deep learning to find out the object is represented by the R-CNN model [8]. Every region proposed from the algorithm is resized into a square and fed into a convolutional neural network from which we extract a 4096-dimensional feature vector as output. Those are now fed into an SVM to classify the presence of the object into the proposed regions. The output vector is also used by a linear regressor to adapt the shapes of the bounding box in order

to reduce the localization errors for every region proposal.

The main problems with the R-CNN are related to the time; in fact:

- The network took a huge amount of time from to get trained;
- Around 47 seconds for every test images, so usless for real time classification.

Fast Region-based Convolutional Network (Fast R-CNN)

Fast R-CNN is a develop of the previous algorithm still developed from Girshick with the aim of reduce the time consumption related to the high number of models necessary to analyse all region proposals is the Fast Region-based Convolutional Network (Fast R-CNN)[9].

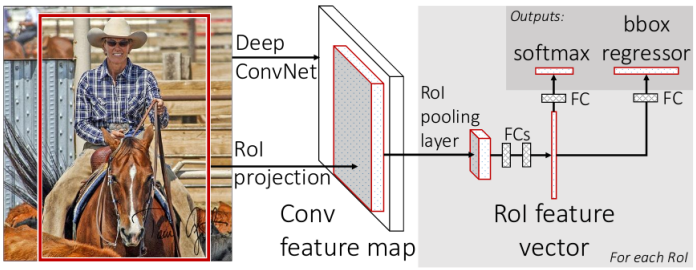


Figure 2.9: Fast R-CNN architecture

Following the architecture in the figure 2.9 a FastR-CNN network takes as input a set of proposed object and an entire image instead of using a CNN for each region proposals (R-CNN). In order to detect the Region of Interests (RoI) a convolutional feature map is produced. For this purpose, the whole image is processed with several convolutional and max-pooling layers. The map's size is then reduced using an RoI pooling layer and a feature vector is extracted. Each feature vector is fed into a sequence of fully connected layers. This is used to predict the observed object with a softmax classifier and to adopt bounding box positions.

Faster Region-based Convolutional Network (Faster R-CNN)

The use of selective search to find out the region proposal is computationally expensive. In order to eliminate this Shaoqing Ren et al.[10] have introduced Region Proposal Network (RPN). This network is able to directly generate

region proposal, predict bounding boxes and detect objects.

As the Fast-CNN a convolutional feature map is produced from the input image. A features vector is obtained as an output of the sliding of a 3x3 window and is connected to two fully-connected layers, one for box-regression and one for box-classification.

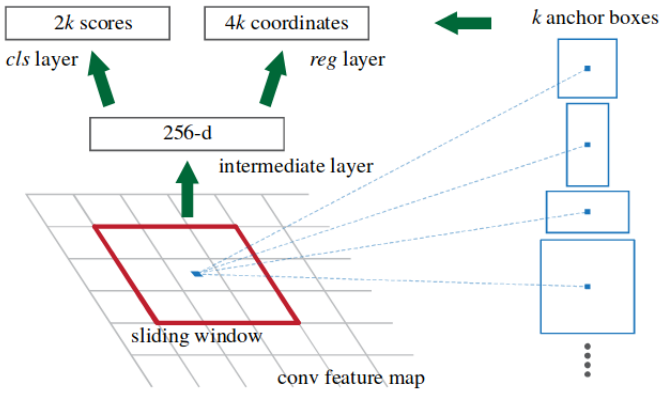


Figure 2.10: Region Proposal Network (RPN)

Multiple region proposals are predicted by the fully-connected layers. The predicted region proposals(archor boxes) are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes. This value is a threshold score to keep only the relevant boxes. A Fast R-CNN model is powered by the archor boxes and the feature maps.

The RPN, to be faster, it is fine-tuned on the PASCAL VOC dataset and use a pre-trained model over the ImageNet dataset for classification. The anchor boxes are used to generate region proposals used to train the Fast R-CNN. All these steps lead to an iterative process2.11.

Region-based Fully Convolutional Network (R-FCN)

Based on the R-CNN model, the R-FCN adopt a two-stage detection strategy. After the region proposal a classification of this region is compute.

The regions are extracted by the RPN. Given them they will be classified from the R-FCN into object catregories and background. All the learnable weight layers in the R-FCN are convulational and are computed in the entire image.

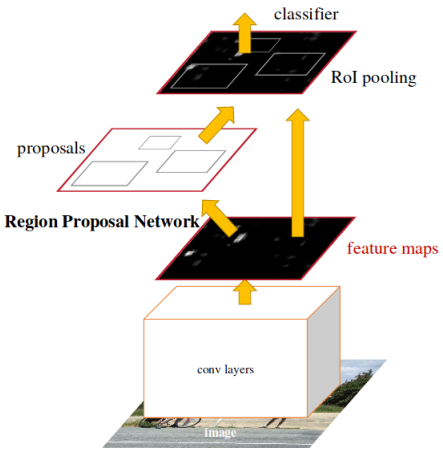


Figure 2.11: Faster R-CNN process

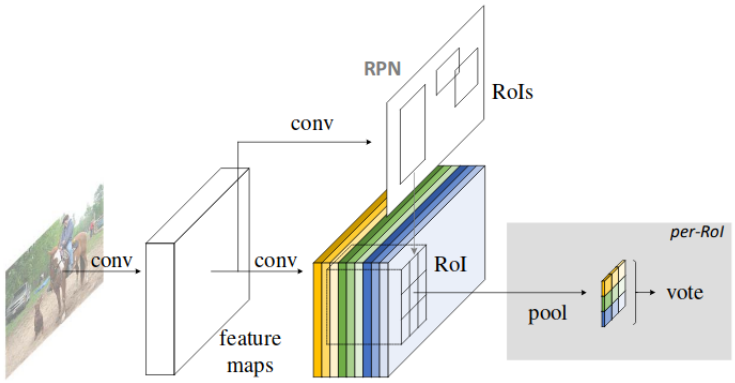


Figure 2.12: Overall architecture of R-FCN

In the last convolutional layer a bank of *"k^2 position-sensitive score maps"* is produced for each category. This bank correspond to a patial grid of *kxk* relative positions. For example, with *kxk*=3x3, the 9 score maps encode the cases of {top-left, top-center, top-right, ..., bottom-right} of an object category2.13. In case that enought relative positions are activated, the vote will be "yes" and therefore the object will be recognized.

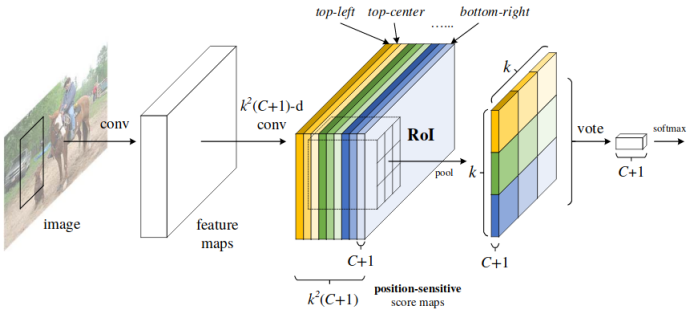


Figure 2.13: Example of R-FCN

Chapter 3

State of the Art

In this chapter are presented the two algorithms that, in the state of the art, are the fastest and most performing object detection algorithm. YOLO(You Only Look Once) and ResNet.

3.1 You Only Look Once

You Only Look Once (YOLO) is a real-time object detection alghoritm[11]. In a single evaluation bounding boxes and classes probability related to them are done by a single neural network from full images.

Its detection performance are directly optimized by itself on full images. Those are due to some aspects:

1. Easy pipeline of work because the detection is a regression problem;

2. Prediction making reasons. Since YOLO sees the entire image during training and test time, not like sliding window and regio proposal-based techniques, it encodes also contextual information about classes and their appearance;

3. Generalizable representations of objects are learned.

Some accuracy problems belonged to YOLO in the first releases are now increased in the latest.

3.1.1 Model

The input image is divided into an *SxS* grid. The grid is responsible for the detection of each object whose centre is into a cell.

Each grid cell predicts B bounded boxes and confidence scores for those boxes. The score is related to how confident the model is that an object is inside and the accuracy of that confidence. We can define it as:

$$Pr(Object) * IOU \,^{truth}_{predict} \qquad (IOU = IntersactionOverUnion[1]) \tag{3.1}$$

The confidence of a no object detection should be zero.

Every bounded box has 5 predictions: *x, y, w, h, confidence*. Centroid coordinates, width, height and confidence prediction(represents the IOU between the predicted box and any ground truth box).

Every grill cell has C conditional class probability conditioned on the fact that contain or not an object.

$$Pr(Class_i|Object) \tag{3.2}$$

Only one set of class probability per grid cell is predicted to look at the number of boxes B.

At test time:

$$Pr(Class_i|Object) * Pr(Object) * IOU \,^{truth}_{predict} = Pr(Class_i) * IOU \,^{truth}_{predict} \tag{3.3}$$

the conditional class probability is multiplied by the individual box confidence predictions which gives a class-specific confidence score for each box.

The result scores shows are how well the predicted box fits the object and the probability of that class appearing in the box(figure 3.1).

3.1.2 Architecture

As already said the model is built as a convolutional neural network. The output prediction about the probabilities and coordinates are developed after the extraction, by the first layers, of features from the image.

24 convolutional layers followed by 2 fully connected layers using *1x1* reduction layers followed by *3x3* convolutional layers.

The final output of our network is the *7 x 7 x 30* tensor of predictions.

[1]Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset.

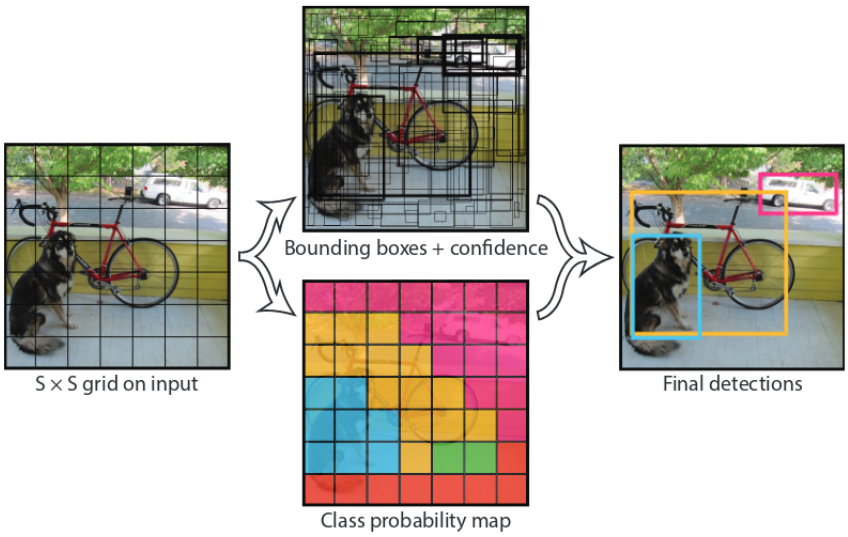


Figure 3.1: YOLO Model.

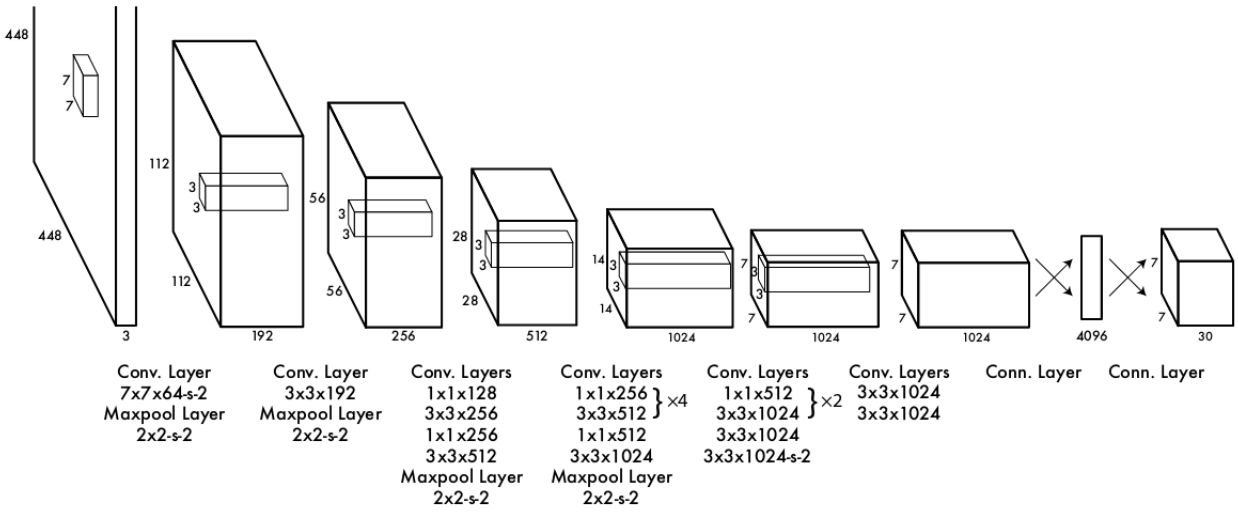


Figure 3.2: YOLO Architecture.

3.1.3 Training

The convolutional layers are pre-trained on the ImageNet 1000-class competition dataset.

The final layer predicts bounding box coordinates and class probability. In order to use low numbers (between 0 and 1), the width and the height are

normalized. The founding box x and y coordinates are parameterized to be offset of a particular grid cell location and so be bounded between 0 and 1.

All the layers, except for the last one that uses a linear activation function, use the following leaky rectified linear activation:

$$\theta(x) = \begin{cases} x, & if x > 0 \\ 0.1x, & otherwise \end{cases} \tag{3.4}$$

The optimization is based on the sum-squared error in the output of the model.

If any object is contained inside a grid cell bring the "confidence" scores towards zero. A frequent consequence is overpowering the gradient from cells that do contain objects. A possible result is model instability, causing training to diverge early on. In order to manage this weakness, the loss is increased from bounding box coordinate predictions and decreased from confidence predictions for boxes that don't contain objects.

YOLO predicts multiple bounding boxes per grid cell, but during the training time, just one bounding box predictor is responsible for each object. The strategy used is based on the highest current IOU with the ground truth, a predictor is "responsible" for predicting an object.

Following the loss function we optimize during the training:

$$\begin{aligned} &\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ &+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ &+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \\ &+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\ &+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2 \end{aligned} \tag{3.5}$$

Where:

- $\mathbb{1}_i^{obj}$: Indicate if an object is in cell i;
- $\mathbb{1}_{ij}^{obj}$: Indicate that the jth bounding box predictor in cell i is "responsible" for that prediction;

If an object is located into a grid cell the loss function penalizes classification error for that cell. Also in a predictor is responsible for a ground truth box the function penalizes bounding box coordinate error.

The network is trained for 135 epochs on the training and validation sets from PASCAL VOC 2007 and 2012 with a batch size of 64, a momentum of 0.9 and a decay of 0.0005.

The learning rate is slowly raise from a rate of 10^{-3} to 10^{-2} and locked for 75 epochs, then 10^{-3} for 30 epochs until 10^{-4} for 30 epochs.

In order to avoid the overfitting, extensive data augmentation and dropout are used. After the first connection layer a dropout layer with rate =.5 prevents co-adaptation between layer. For the data, augmentation is used random scaling and translations of up 20% of the original image size. Exposure and saturation of the image are randomly adjusted.

3.2 Residual Network

Residual Network (ResNet)[12] is a deep neural network based on the reformulation of the layers as learning residual functions with reference to the layer inputs. It aims to ease the training of the network.

The main problem of the deeper networks is that when they are able to start converging, a *degradation* problem shows up. So on the increasing of the depth, accuracy gets saturated and then degrades rapidly.

The algorithm introduces a deep residual neural network in order to correct the degradation problem. For this purpose, each layer is lead to fit a desired residual map.

In order to show the degradation problem and evaluate ResNet, we can say that the network:

1. is easy to optimize but higher training error when the depth increase;

2. can experience accuracy gains from the depths increase and producing good results;

3.2.1 Model

Consider an underlying mapping $H(x)$ to be fit by a few stacked layers with x as inputs to the first of this layer.

If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., $H(x)x$ (assuming that the input and output are of the same dimensions). So rather than expect stacked layers to approximate $H(x)$, we explicitly let these layers approximate a residual function $F(x) := H(x)-x$. The original function thus becomes $F(x) + x$. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different.

In the case of optima identity mapping, in the residual learning reformulation, the solvers can bring the weights of the multiple nonlinear layers toward zero to approach identity mapping.

The residual learning is adopted every few stacked layers with building blocks(figure 3.3) defined as:

$$y = F(x, \{W_i\}) + x \tag{3.6}$$

With:

- x and y: input and output vectors of the layers considered;
- $y = F(x, \{W_i\})$: function that represents the residual mapping to be learned;

x and F must have the same dimension, otherwise we can perform a linear projection W_S by the shortcut connection to match the dimensions:

$$y = F(x, \{W_i\}) + W_S x \tag{3.7}$$

The residual function F has a flexible form.

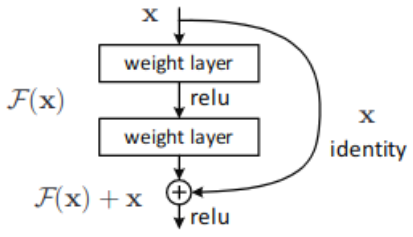


Figure 3.3: ResNet building block.

Plain Network

The convolutional layers mostly have 3x3 filters and follow two simple design rules:

1. for the same output feature map size, the layers have the same number of filters;
2. if the feature map size is halved the number of filters is doubled so as to preserve the time complexity per layer.

The downsamping is performed directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layer is 34.

Residual Network

Based on the above plain network, shortcut connections are inserted(Figure 3.4) they turn the network into its counterpart residual version. The identity shortcuts (Eqn 3.6) can be directly used when the input and output are of the same dimensions. When the dimensions increase two options are considered:

> [(A)]The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; The projection shortcut in (Eqn 3.7) is used to match dimensions(done by *1x1* convolutions).

For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

3.2.2 Implementation

The image is resized with its shorter side randomly sampled in [256, 480] for scale augmentation. A 224x224 crop is randomly sampled from an image or its horizontal flip, with the pre-pixel mean subtracted. The standard colour augmentation is used. Batch normalisation (BN) is adopted right after each convolution and before activation. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus and the models are trained from up to $60x10^4$ iterations. The weight decay is 0.0001 and momentum is 0.9. We do not use dropout.

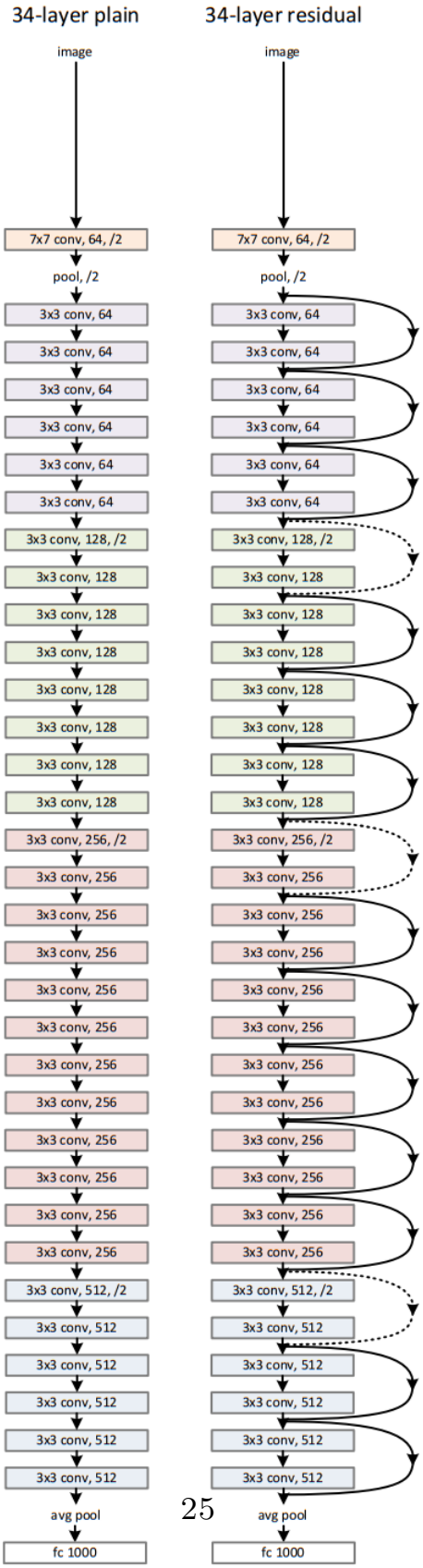


Figure 3.4: Architectures for ImageNet.

Chapter 4

Robot Operating System

Robot Operating System (ROS) is a flexible framework for writing robot software. It is an open-source, meta-operating system for robots that aims to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

4.1 Characteristic

It provides services like hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. A set of processes inside of ROS can be represented in a graph as nodes that can receive, send and multiplex messages coming and direct to other nodes, sensors, and actuators. Even if the low level of latency of the operations to control a robot, ROS is not a real-time operating system, but it is even possible to integrate it with some real-time modules.

4.2 Components

In the ROS architecture software can be grouped into three categories:

1. Instruments to develop a publishing software based on ROS;
2. Libraries for ROS client processes like roscpp, rospy and roslisp;
3. Packages holding applications and codes that use one or more libraries for ROS client processes;

4.2.1 Ros Visualization

RVIZ is a ROS graphical interface, a three-dimensional visualizer used to visualize robots, the environments they work in, and sensor data. It allows users to visualize a lot of information, using plugins for many kinds of available topics.

4.2.2 Transform Library

TF plugin allows visualizing the positions and the orientations of all the frames that compose the TF Hierarchy.

TF is a package that lets the user keep track of multiple coordinate frames over time. TF maintains the relationship between coordinate frames in a tree structure buffered in time and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

Key parameters:

[*]Show names: Enable/disable the 3D visualization of the name of the links Show axes: Enable/disable the 3D visualization of the axes of the frames Show arrows: Enable/disable the 3D visualization of the arrows that connect the various frames

4.3 Programming languages

The libraries and the development tools are independent on the programming language used(C++, Python e LISP) and are realised under license BSD. There are free software for commercial and research use. The biggest parts of the packages are realised under different open source licenses. Those packages are used for functionalities like: drivers, three-dimensional robot model, 2D and 3D simulation, image visualisation. localisation instruments, maps and other algorithms.

Chapter 5

Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) problem aim of constructing or updating a map of an unknown environment while simultaneously keeping track of a mobile robot location within it.

It's a multi-stage process that includes alignment of sensor data using a variety of algorithms well suited to the parallel processing capabilities.[13]

The "solution" of the SLAM problem has been one of the notable successes of the robotics community over the past decade. SLAM has been formulated and solved as a theoretical problem in many different forms. SLAM has also been implemented in some different domains from indoor robots to outdoor, underwater, and airborne systems.

At a theoretical and conceptual level, SLAM can now be considered a solved problem. However, substantial issues remain in practically realising more general SLAM solutions and notably in building and using perceptually rich maps as part of a SLAM algorithm.

5.1 Problem definition

The figure 5.1 shows a mobile robot moving to an environment taking relative observations of several unknown landmarks using a sensor located on the robot.

At a time instant k, the following quantities are defined:

3. x_k: the state vector describing the location and orientation of the vehicle;

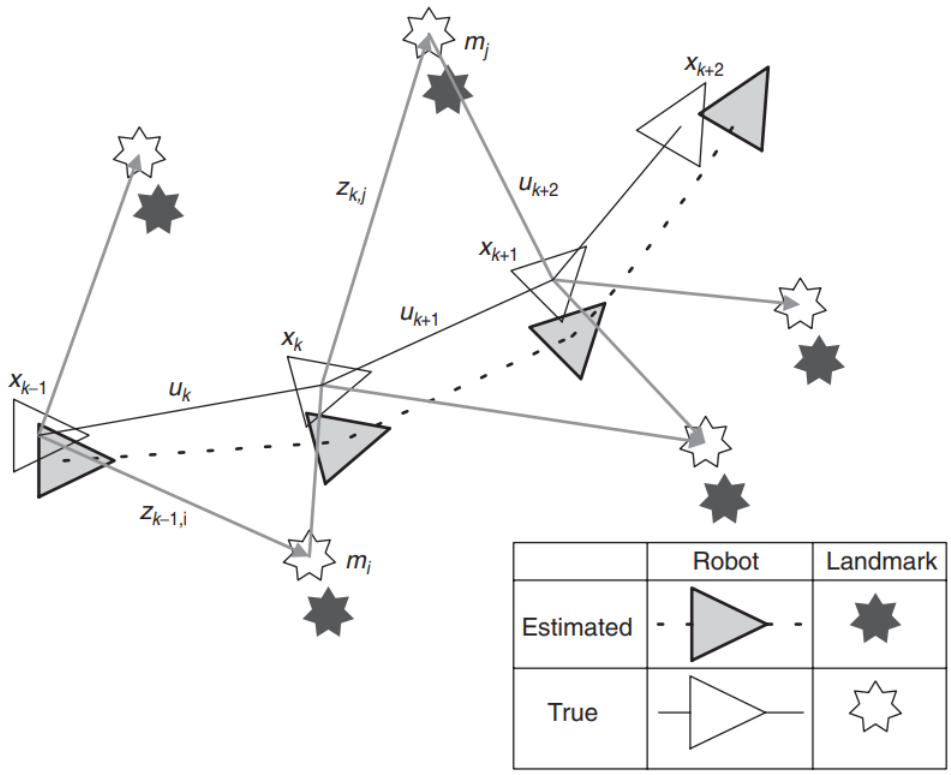


Figure 5.1: SLAM problem.

2. u_k: the control vector, applied at time $k-1$ to drive the vehicle to a state x_k at time k;

3. m_i: a vector describing the location of the ith landmark whose true location is assumed time invariant;

4. z_{ik}: an observation taken from the vehicle of the location of the ith landmark at time k. When there are multiple landmark observations at any one time or when the specific landmark is not relevant to the discussion, the observation will be written simply as z_k.

In addition, the following sets are also defined:

1. $X_{0:k} = \{x_0, x_1, ..., x_k\} = \{X_{0:k-1}, x_k\}$: the history of vehicle locations;

2. $U_{0:k} = \{u_1, u_2, ..., u_k\} = \{U_{0:k-1}, u_k\}$: the history of control inputs;

3. $m = \{m_1, m_2, \cdots, m_n\}$ the set of all landmarks

4. $Z_{0:k} = \{z_1, z_2, \cdots, z_k\} = \{Z_{0:k-1}, z_k\}$: the set of all landmark observations.

5.2 Probabilistic SLAM

This solving method of SLAM requires that:

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{5.1}$$

the probability distribution(5.1) is computed for all times k. The equation (5.1) describes the joint posterior density of the landmark locations and vehicle state (at time k) given the recorded observations and control inputs up to and including time k together with the initial state of the vehicle.

The join posterior, is computed using Bayes theorem, after an estimation for the distribution $P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1})$ at time *k-1* following a control u_k and observation z_k.

Describing the effect of the control input and observation we define the state transition model and an observation model required from the computation above. The probability of making an observation z_k is described by the **observation model**:

$$P(z_k|x_k, m) \tag{5.2}$$

when the vehicle and landmark locations are known. It reasonable assume that observations are conditionally independent once the vehicle location and map are defined, given the map and the current vehicle state.

The probability distribution on state transitions can describe the **motion model**:

$$P(x_k|x_{k-1}, u_k) \tag{5.3}$$

for the vehicle. Where the state transition is assumed to be a Markov process in which the next state x_k depends only on the immediately proceeding state x_{k-1} and the applied control u_k, and it is independent of both the observations and the map.

5.2.1 Structure of Probabilistic SLAM

The dependence of observations on both the vehicle and land-mark locations are made explicit from the observation model (5.2).

The joint posterior can not be partitioned as:

$$P(x_k, m|z_k) \neq P(x_k|z_k)P(m|z_k) \tag{5.4}$$

and is well known, from previous research on consistent mapping [14] [15], that a partition similar to this leads to inconsistent estimates.

As we can see from the figure 5.1, much of the error between estimated and true landmark locations(errors of where the robot is when landmark observations are made) is common between landmarks and it is because of a single source.

This implies that the errors in landmark location are highly correlated, basically means that the relative location between any two landmarks, $m_i - m_j$ may be known with high accuracy, even when the absolute location of a landmark m_i is quite uncertain. From the probabilistic point of view, this means that the joint probability density for the pair of landmarks $P(m_i - m_j)$ is highly peaked even when the margin densities $P(m_i)$ may be quite dispersed.

It is known that the correlations between landmark estimates increase monotonically as more and more observations are made(for the linear Gaussian case)[16]. So we can see that the knowledge of the relative location of landmarks, for robot motion, never diverges but always improves. From a probabilistic view, the joint probability density on all landmarks $P(m)$ becomes monotonically more peaked as more observations are made.

Referring again to Figure 5.1, if we take a look at the relative location of observed landmarks is clearly independent of the coordinate frame of the vehicle and successive observations from his fixed location. This considers the robot at location x_k observing the two landmarks m_i and m_j.

Assuming now the robot moves to the location x_{k+1}, if it observes the landmark m_j for a second time, it will lead the estimated location of the robot and landmark to be updated relative to the previous location x_k. This last action will propagate back to update the landmark m_i even if is not seen any more from the new location. This is due by the high correlation between the two landmarks from the previous measurements and the update increases again the correlation between those two.

Every time we move to a new location there is the possibility to increase the observation of new landmarks immediately correlated or linked to the rest of the map. This new update will bring cascade updates to the previous landmarks.

We can see from Figure 5.2 what we obtain from this process. In this network, the springs(the connection between landmarks) become increasingly (and monotonically) stiffer as the robot moves through this environment and takes observations of the landmarks.

At the end of this process an accurate relative map of the environment or

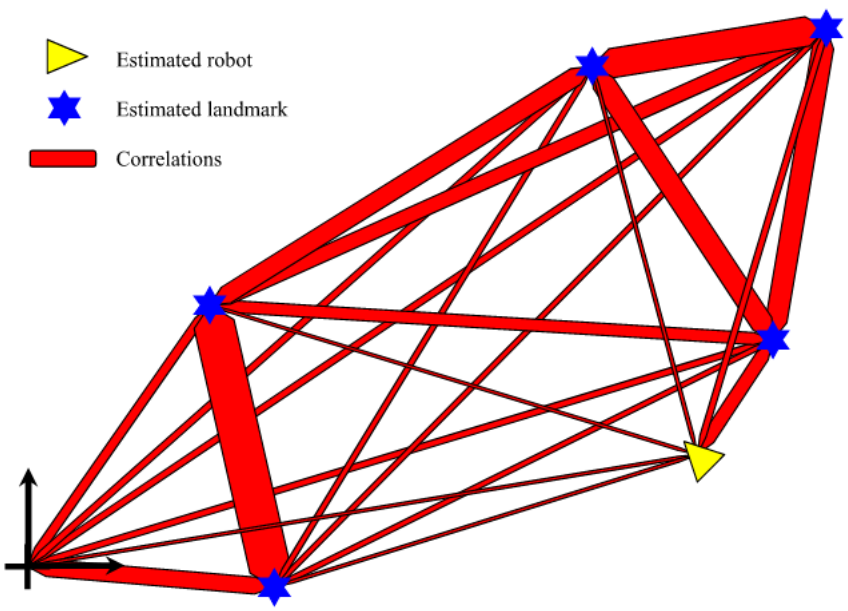


Figure 5.2: Landmarks network.

a rigid map of landmarks is obtained.

> In the theoretical limit, robot relative location accuracy becomes equal to the localisation accuracy achievable with a given map

The location accuracy of the robot is bounded by the quality of the map and relative measurement sensor.

5.2.2 Solution to the probabilistic SLAM

Solutions to the probabilistic SLAM are made by finding feasible representation for the observation model Equation 5.2 and motion model Equation 5.4.

> [13] One important alternative representation is to describe the vehicle motion model in Equation 5.4 as a set of samples of a more general non-Gaussian probability distribution. This leads to the use of the Rao-Blackwellised particle filter, or Fast-SLAM algorithm, to solve the SLAM problem as described in Section IV-B. While EKF-SLAM and FastSLAM are the two most important solution methods, newer alternatives, which offer much potential, have been proposed including the use of the information-state form[17]

5.3 Graph-based SLAM

> A graph-based SLAM approach constructs a simplified estimation problem by abstracting the raw sensor measurements.[18]

The raw measurements(virtual measurements) are replaced with the edges in the graph. The probability distribution over the relative locations of the two poses labels the edge between two nodes, conditioned to their mutual measurements. In general, the observation model $p(z_t|x_t, m_t)$ is multi-modal and therefore the Gaussian assumption does not hold. This means that a single observation z_t might result in multiple potential edges connecting different poses in the graph and the graph connectivity needs itself to be described as a probability distribution. A combinatorial complexity could be reached due to the multi-modality in the estimation process.

As result, most practical approaches restrict the estimate to the most likely topology, introducing the problem of data association because we need to determine the most likely constraint resulting from observation and this depends on the probability distribution over the robot poses. This problem is usually addressed by the front-end SLAM. To compute the correct data-association is required a consistent estimate of the conditional prior over the robot trajectory $p(x_{1:T}|z_{1:T}, u_{1:T})$.

During the computation, some observations can be affected by Gaussian noise. In this case, and the data association is known, the goal of the graph-based mapping algorithm is to compute a Gaussian approximation of the posterior over the robot trajectory by computing the mean of this Gaussian as the configuration of the nodes that maximises the likelihood of the observations.

Let's take a vector x of poses of each node i, $x = (x_1, ..., x_n)^T$. Then the mean z_{ij} and the information matrix of a virtual measurement between the node i and the node j, Ω_{ij}. With Ω_{ij} as a transformation that guarantees that the observations acquired from i maximally overlap with the observation acquired from j. Let $\hat{z}_{ij}(xi, xj)$ be the prediction of a virtual measurement given a configuration of the nodes x_i and x_j.

The log-likelihood l_{ij} of a measurement z_{ij} is:

$$l_{ij} \propto [z_{ij} - \hat{z}(x_i, x_j)]^T \Omega_{ij} [z_{ij} - \hat{z}_{ij}(x_i, x_j)] \tag{5.5}$$

Let $e(x_i, x_j, z_{ij})$ be a function that computes a difference between the expected observation $\hat{z}_{ij}$ and the real observation z_{ij} gathered by the robot.

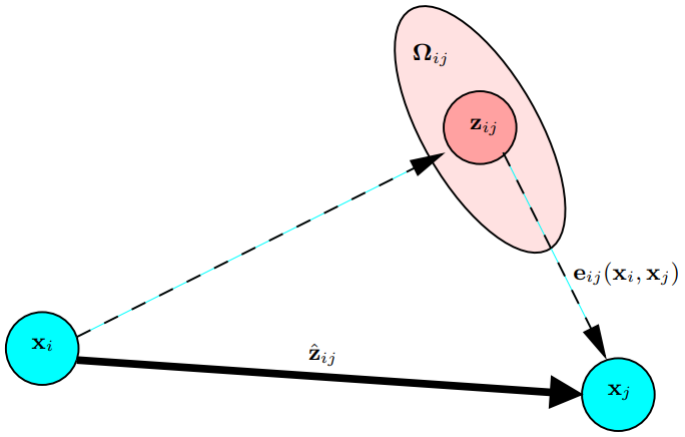


Figure 5.3: Aspects of an edge.

The Figure 5.3 shows the aspects of an edge connecting the vertex xi and the vertex x_j . This edge originates from the measurement z_{ij}. From the relative position of the two nodes, it is possible to compute the expected measurement $\hat{z}_{ij}$ that represents x_j seen in the frame of x_i. The error $e_{ij}(x_i, x_j)$ depends on the displacement between the expected and the real measurement. An edge is fully characterized by its error function $e_{ij}(x_i, x_j)$ and by the information matrix Ω_{ij} of the measurement that accounts for its uncertainty. Let's introduce now $\mathcal{C}$ as the indices's set of pairs where z exists. We will minimizes the negative log likelihood F(x) by finding the configuration of the nodes x^* using a maximum likelihood approach.

$$F(x) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{F_{ij}} \tag{5.6}$$

it tries to solve:

$$x^* = \underset{x}{argmin} F(x) \tag{5.7}$$

In order to solve 5.7 can use different approaches utilizes standard optimization methods, like the Gauss-Newton or the Levenberg-Marquardt algorithms, it is particularly efficient because it effectively exploits the structure of the problem.

Chapter 6

Tracker

The target deals with the research of an algorithm able to compare two (or more) images and to determine if one of them is a duplicate (the same or similar) of the other one, in order to keep only distinct images into a database. There are three strategies to fix this problem:

- The first method consists to choose the best 100 pixels, to compare images basing only on them. The "best pixels" means pixels that provide the most important information about the image, generally borders or edges. Today the commonest key points are Scale-Invariant Feature Transform (SIFT), because they are efficient even if the image to compare is rotated, scaled, or differently lit. This approach, even if it is smart, is also so much wasteful, in fact, it costs $T(n) = O((n^2)m)$, where n is the number of image's key points and m is the number of images stored in the database. There are some algorithms that try to reduce the cost, for example with faster identification of the closest correspondence. Two examples of them can be the "Quadtrees" and "Binary space partitioning";

- The second method, called "histogram method", consists to create functionality histogram for each image, comparing them with ones generated by the input image, to choose one that has more correspondence. It is faster than the first method, but less solid, because it is efficient only for images that are similar to ones on the database. Nonetheless, it is not unreliable because small changes, as a small cutout, does not affect the functionality of the algorithm. It is possible to implement that for example with five histograms, three of colours and two of weft (direction and scale).

- The third method consists to use semantic text forests (PDF) to extract simple key points and to classify images by collection decision trees. This approach is faster than the first one, because it avoids the matching process and because key points are easier than SIFT, therefore also the extraction is faster than the first one. It is also more solid than the second method, because it works even If images are rotated, scaled, or differently lit.

In this case, to make the algorithm more solid, two techniques have been used: the first that finds key points and descriptors with SURF algorithm and that correlates them to each other with FLANN algorithm and the second one that uses the "Perceptual Hash" technique. The two techniques are described, in particular, below:

6.1 Speeded-Up Robust Features and Fast Library for Approximate Nearest Neighbors

This technique plans to apply the SURF algorithm to find key points and descriptors of the image and, after that, to apply FLANN method that, with the calculation of distances, chooses matches that have less distance, that are ones with the best correspondence into the dataset.

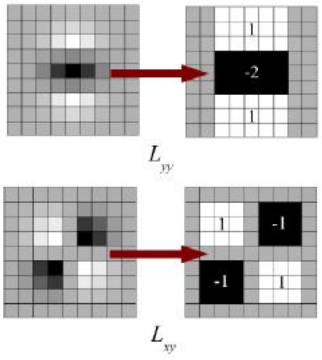


Figure 6.1: Surf algorithm operation.

In 2006, to fix the problem of poor speed of the SIFT algorithm, Bay, H., Tuytelaars, T. and Van Gool published a new one, called SURF, that was an update and improved version of the first one. It resulted three times faster than SIFT, but keeping performance already achieved. Unlike SIFT, SURF approximates the "Laplacian of Gaussian" with Box Filter, to calculate the

convolution with the help of integral images and to allow to calculate them in parallel for different scales. As the first algorithm, the second relies on the determinant of the hessian matrix for the scale but also for the position. Therefore, we can say that SURF is affordable to handle hazy and rotated images, but it is less efficient for a different point of view and different lighting. Some functionalities of the algorithm are described below:

- Orientation:

  For some applications, the invariance of rotation is not required, so in those cases it is not necessary to assign the orientation, speeding up the process. However, when it is required, wavelet answers are used (discoveries through integral images on any scale) in horizontal e vertical direction, for a neighbour of 6s dimensions, to which relevant Gaussian weights are applied. Dominant orientation is estimated calculating the sum of all the answers into a sliding orientation window with an angle of 60 degrees.

- Functionality descriptor:

  SURF uses wavelet answers in a horizontal and vertical direction. It is taken a neighbourhood 20s x 20s, divided into sub-regions 4 x 4. For every sub-region wavelet, answers are taken and a vector is formed. It provides a SURF functionality descriptor with 64 total dimensions. $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$. The calculate (and correspondence) speed increases reducing dimensions, but in this way, it is provided with the best distinctive character of the characteristics. Increasing dimensions, for example to 128, sums dx and $|dx|$ are calculated in a different way according to the sign of dy (the other way around for dy). This case does not increase computational cost, even if it doubles up the number of functions.

- Below points of interest:

  It is used the Laplacian sign (track of Hessian Matrix), which has already been calculated during the discovery (so it doesn't add calculation cost). It allows distinguishing bright patches on dark backgrounds and dark patches on bright backgrounds, to compare only images that have the same kind of contrast. In this way, we have faster correspondence, without affecting descriptor performances.

6.2 FLANN algorithm functionality

Identified key points and descriptors, to correlates them, the concept of "nearest neighbour search problem" has been defined and used for the elaboration of points of a vector space, to find (using an efficient way) points close to a new query point. For hierarchical k-means tree, the method that has been used is a priority queue to expand the research in order according to distances of every k-mean domain of the query. While the method that is usually known uses a branch-and-bound approach, this one searches deep in the first order[19]. The algorithm that is most frequently used for the search of the nearest neighbour is the kd-tree (Freidman et al., 1977), that is efficient for low size, but it quickly loses its effectiveness with the increase of dimensions. Over time a lot of algorithms have been proposed and modified, to improve speed and effectiveness. Those that seem more efficient are ones that use hierarchical trees or randomized kd-trees, because the accuracy of the approximation is measured in terms of precision of question marks for which the nearest neighbour is located.

- The randomized kd-tree algorithm

  The original kd-tree algorithm splits the data at each level of the tree in half, over the dimension for which the data show greater variability. It results efficient for low dimensions, but performances rapidly worsen with large size. To obtain a speedup on the linear search becomes necessary been satisfied with a closest approximate neighbour, because it increases search speed, but it is not so accurate. To fix this problem Silpa-Anan e Hartley proposed an improved version, in which are created more randomized kd-trees, that are built choosing the dimension randomly divided among the first dimensions on which the data have greater variability. The approximation degree is determined by examining a landline number of leaf knots, then the search is closed, and the best containers are returned. In this implementation, the user-specified only the search precision desired.

- The hierarchical k-means tree algorithm

  In this case, the tree is built dividing data points on each level into K distinct regions using a grouping of k-means and then applying the same method in a recursive way to points of each region. The recursion ends when the number of points in a region in less than K. *Figure 1 shows projections of different hierarchical k-means trees built with

SIFT functionality with 100 K, using different factors of ramification* In reality, the implementation used tries to improve the exploration of the kd-tree, performing a single crossing and adding a priority queue all unexplored branches in each node during the way. After that, it extracts from priority queue the branch that has the centre closest to query point and restarts the crossing from that branch. The approximate degree is specified in the same way of randomized kd-tree, with the interruption of the search after a predetermined number of leaf nodes have been examined.

- Automatic selection of the optimal algorithm

  In reality, the selection of the optimal algorithm for a fast approximative search of the closest neighbour depends on different factors, for example, the data set structure or the search precision desired. There are a series of parameters, for each algorithm, those influence performances, for example, the number of randomized trees in kd-trees case, or the ramification factor in case of hierarchical k-means tree. So, we can say that the research of the best algorithm is reduced to the research of the best parameters, which make the algorithm optimal. It is also possible to decide whether to apply the optimization to the complete set of data or only to a fraction of it. The first option provides the best result, obviously, but it has a longer execution time, while the second one is faster and it is equally effective if the data set is chosen appropriately, for example, a tenth of the total.

6.3 Perceptual Hash

The perceptual hash algorithm plans to reduce the image to a small hash code, as if it was a kind of fingerprint, to use a more compact representation (and consequently a comparison). While cryptographic hash functions are based on the avalanche effect of small input changes leading to drastic changes of the output, the perceptive hashes are “close” to each other if characteristics are similar. They must be enough robust to cope with transformations or modifications of input, but at the same time, they must be enough flexible to distinguish between different files. These changes can be rotations, inclinations, regulation of contrast and different formats/compressions. The generally used algorithm is “Average hash”, bus in this case “PHash” has been used because it is slower than the first one, but it tolerates better modifies, so it is more reliable. This kind of algorithm use functions, called “hash

functions", that can be classified in "unkeyed hash functions" and "keyed hash functions". First of them generate a random variable from a random input, while second ones generate random variable from an input that contains a secret key. PHash uses ones of the first type, which are, in specific:

- Discrete Cosine Transform based hash:

  It calculates fixed length hashes (64 bits), and it saves them in a string. It plans to calculate the mean of the 64 coefficients DCT, and then to normalize the sequence in a binary form, to obtain the final hash value:

  $$h = \begin{cases} 0, & if \quad Ci < m \\ 1, & if \quad Ci \geq m \end{cases} \qquad (6.1)$$

  where hi is the perceptive image's hash bit in i position.

- Radial variance-based hash:

  It calculates fixed length hashes (320 bits) and it saves them in an away uint8 t (coeffs). To determine the final hash, Peaks of Cross-Correlation (PCC)) between the two hash values is calculated, and it is compared with a variable threshold value. If the PCC is under the threshold, images are considered the same, otherwise, they are different. Radial variance-based hash is the only one that does not normalize the image compared to resolution.

- Marr-Hildreth operator-based hash:

  It calculates fixed length hashes (576 bits) and returns a pointer to the string that contains the hash. Unlike the other functions, this one implements the calculate of hamming distance normalized for a kind of hash, but before the extraction of the image, it is pre elaborated. It is subjected to a conversation to greyscale, it is blurred and resized. At the end of the process, an equalized version of the image histogram is calculated, using 256 levels of the histogram.

Recently, a perceptive hash function has been inserted, which is based on average blocks values, where the hash is returned into a BinHash object.

Chapter 7

Ground Robot - HUSKY

The robot shown in figure 7.1 is a composition of different sub part that we will analyse in the next chapters. The base is an Husky UGV outdoor robot. On the top of it we have the UR5e, a flexible collaborative robot arm and a Depth Camera D435.

Figure 7.1: Ground robot.

7.1 Ground Vehicle

Figure 7.2: Husky.

Husky (fig. 7.2) is an unmanned ground vehicle. It is a medium-size rover with a development platform. The power system and the large payload capacity allow it to accommodate a variety of different payloads useful to be customized depending on the use situation. Many different sensors can be added like GPS, IMUs, Stereo cameras, LIDAR, etc.. It can rely on the high-torque drivetrain and rugged construction. The UGV has full support in ROS with open source code.

Husky is born with fully ROS support, within a big, growing community. It uses a serial open-source protocol with API support for ROS, and options for C++ and Python. Its high-performance, maintenance-free drivetrain and large lug-tread tires allow Husky to tackle challenging real-world terrain.

Various research papers have been published using Husky as the test set-up. Husky's community is made by hundreds of researchers and engineers globally. Husky is a proven benchmark for all kind of robot research.

Husky is plug-and-play compatible with a wide range of robot accessories. The controller offers smooth motion even at slow speeds (<1cm/s) and with excellent disturbance rejection.

7.2 Robot Arm

The UR5e (fig. 7.3) is a lightweight 6-axis robot arm with long-term flexibility and it is a popular cobot for industry produced by Universal Robots. A leader in collaborative robots.

Figure 7.3: UR5e.

Some specifications:

- Automates tasks with payload up to 11 lbs (5 kg)
- Reach radius of up to 33.5 in (850 mm)
- Weighs only 40.6 lbs (18.4 kg)
- Small footprint of just 5.9 inches (149 mm)
- 6-axis motion with 360-degree wrist-joint rotation and infinite end-joint rotation

The arm lends itself very well to optimize collaborative processes such as pick-and-place, machine tending, and testing applications also because it is easy to program and redeploy, and programs can be reused for recurrent tasks.

It is mounted on the top and brings to the robot the capacity of pick-up objects.

The UR5e feature precise 0.1mm repeatability. It is safe to work alongside human workers without any other secure.

7.3 Depth Camera

Figure 7.4: Intel® RealSense™ depth camera D435.

This is the main component we mostly used and at the center of all the developed algorithms.

The Intel® RealSense™ depth camera D435 (fig. 7.4) is a stereo solution that offers good image quality indoor and outdoor. It perfectly fits with a big variety of applications such as robotics or augmented and virtual reality.

The camera has a cross-platform support that helps it to be integrated into any solution. It has a rage up to 10m with an image sensor technology Global Shutter, 3μm x 3μm pixel size. The accuracy can depend on scene, calibration and lighting condition. In the table 7.1 is shown a list of specifications.

Depth	**Depth Technology:** Active IR Stereo	**Minimum Depth Distance (Min-Z):** 0.105 m
	Depth Field of View (FOV): 87°±3° x 58°±1° x 95°±3°	**Depth Output Resolution and Frame Rate:** Up to 1280 x 720 active stereo depth resolution. Up to 90 fps.
RGB	**RGB Sensor Resolution and Frame Rate:** 1920 x 1080	**RGB Sensor FOV (H x V x D):** 69.4° x 42.5° x 77° (+/- 3°)
	RGB Frame Rate: 30 fp	
Major Components	**Camera Module:** Intel RealSense Module D430 + RGB Camera	**Vision Processor Board:** Intel RealSense Vision Processor D4
Physical	**Form Factor:** Camera Peripheral	**Connectors:** USB-C* 3.1 Gen 1*
	Length x Depth x Height: 90 mm x 25 mm x 25 mm	**Mounting Mechanism:** One 1/4-20 UNC thread mounting point.Two M3 thread mounting points

Table 7.1: Camera specifications.

Chapter 8

Method

In this chapter is described how the work was actually carried out. All the sources are available from `https://gitlab.liu.se/lrs/lrs_semantic_mapping`.

8.1 Object Distance

The main idea behind this code is something that allows combining the object detection algorithms explained before with SLAM. We take datas coming from the object detection and we produce observations that can be read by the SLAM process.

8.1.1 Pre-study

We already explained what is and what was the aim of our code. In the beginning, we faced the issue and we tried to explore it in order to develop something concrete and robust. We started by analysing which components we had. The robot has, as we already explained in chapter 7, a series of sensors. Our approach was to decide which one of them we could use to help to develop our goal. We choose to build the system around the camera sensor because it is also used from the object detection algorithms.It is composed, in fact, from two cameras:

1. Image colour camera;
2. Black and White depth camera.

This means that we can use the same images of the algorithm process, so, the same objects that we receive from them as ROS messages. These sharing

images are really useful because are also combined with the depth camera's measures.

After this first step in which we analyse the hardware components we focused on the software part: We started from the object detection; we analyzed the algorithms to choose which one to use.

Based on the searching made on the state of the art of object detection, the real-time object detection algorithms we choose were:

- YOLO
- ResNet

The details of them are already explained in the theory chapter n. 3.

In order to implement them on our system we used those developed ROS package:

- DarkNet YOLO
- Tensorflow object detection - ResNet

The first one allows using YOLO (V3) on GPU and CPU[20]. The pre-trained model of the convolutional neural network is able to detect pre-trained classes including the data set from VOC and COCO, or it also possible to create a network with our own detection objects. The second one allows using one of 28 models from Tensorflow models repository[21]. The one we used from this repository was ResNet that allows using the algorithm in a ROS node. Both nodes publish a message(a bit different for each one) containing the bounding boxes, the label (membership class) and the probability of each object to actually belong to the Class.

Then we started planning our algorithm. This step was divided into two sub-steps:

1. Object distance ad feature extractor;
2. Object tracker.

With the first one, we built something that allows us to measure the distance of each object from the camera and extracts some other parameters useful for the other part. After that, we built the tracker, whose aim is to recognize the same object in different frames. This technology has a structure to store the objects we collect from the cameras and check if these objects match with the one just captured by the same. We will better show the technologies we used in the next chapter.

8.1.2 Implementation

After the pre-study phase in which we decided some guidelines for the algorithm, we started with the implementation.

The algorithm is built as a ROS node that is a subscriber to the channels that publish these contents:

- Camera information;
- Colour images from the camera;
- Black and white images from the camera;
- Bounding boxes from the object detection algorithms;

We made two different versions of the same algorithm, due to the different ROS message published from the object detection nodes(YOLO, ResNet).

We started describing the support structure, as we mentioned in the previous chapter, to store the tracking objects. This structure contains different fields. The ID of the object is recognized between frames. In this way, if the ID or the image (even a crop from the full) is seen more than one time, it will be represented as the last time we analyzed that, and the same label(that object detector algorithm assigned at the crop before) will be assigned to it. Other useful information are the "centre" of the object(basically the coordinate expressed in pixels of the centre of the object inside the image) and the distance between the camera and the objects detected. We calculated the distance with the combination of the black and white frame and the object's centre(Fig 8.1).

At this point we knew what we were looking for, so we focused on how data were taken and especially how they were elaborate.

We already said that we can divide the main idea into two sub-blocks. The first one aims to get the times, the images and the distance, from the information derived from channel we are subscribed to. The second block tries to recognize if the object we are processing it is already inside the structure or is a new one. We can call this second block as object tracker. In order to solve the last problem described, we used two different methods of image comparison: *SURF* combined with *FLANN* and *PHash*. We already described the two used algorithms in Chapter n. 6. The reason why we implemented both of them is that we wanted to have a robust tracker for the images.

Assuming that the second technique is better than the first, if at the end of the analysis the latter gives a result, this is considered as excellent. While

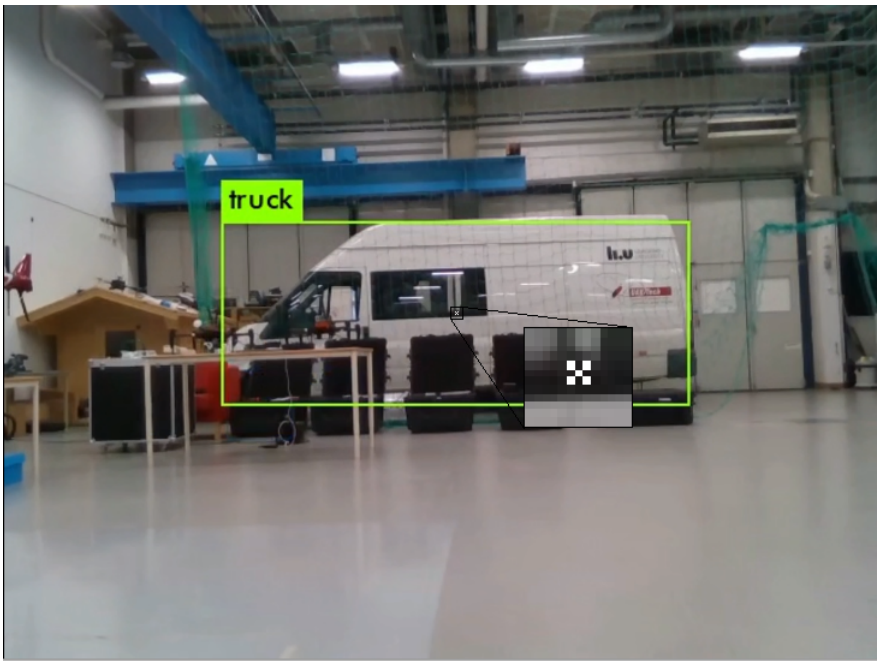


Figure 8.1: The object's centre and the label of it. The centre is represented by the effective centre and other pixels taken as shown in the picture(white pixels). This choice lead to more accuracy for the distance calculation.

if no result has been found, the solution provided by the first technique is used (in case this has found one). This allows us to have a double control that guarantees greater robustness and efficiency to the program and gives to the tracker a good accuracy in terms of a past object correctly recognize inside the new frames. All the data collected are publish as a ROS message(Observations message) on the channel *"/darknet/observations"* for the darknet and *"observations"* for the ResNet.

We will see later some tests done on the algorithm in order to prove is strengths and weaknesses.

8.1.3 Evaluation

As we said some tests are conducted in order to evaluate the work done here. We will just list them. The discussions are postponed to the next chapter(Chapter n. 9). Tests:

- Test1: Label accuracy;
- Test2: Reliability of detection;

- Test3: Tracker;
- Test4: Object distance;

8.2 Vector Pose Observation

We wanted to show with a graphical representation the distance between the object and the camera of the robot.

8.2.1 Implementation

We developed a small tool to have a graphical representation of what the robot sees as distance between him and the object that he frames with the camera. It works with the ROS RVIZ tool (Chapter n. 4). As shown in the pictures (8.2 and 8.3) we can see the representation of the distance between the centre of the object and the camera of the robot in the form of arrows.

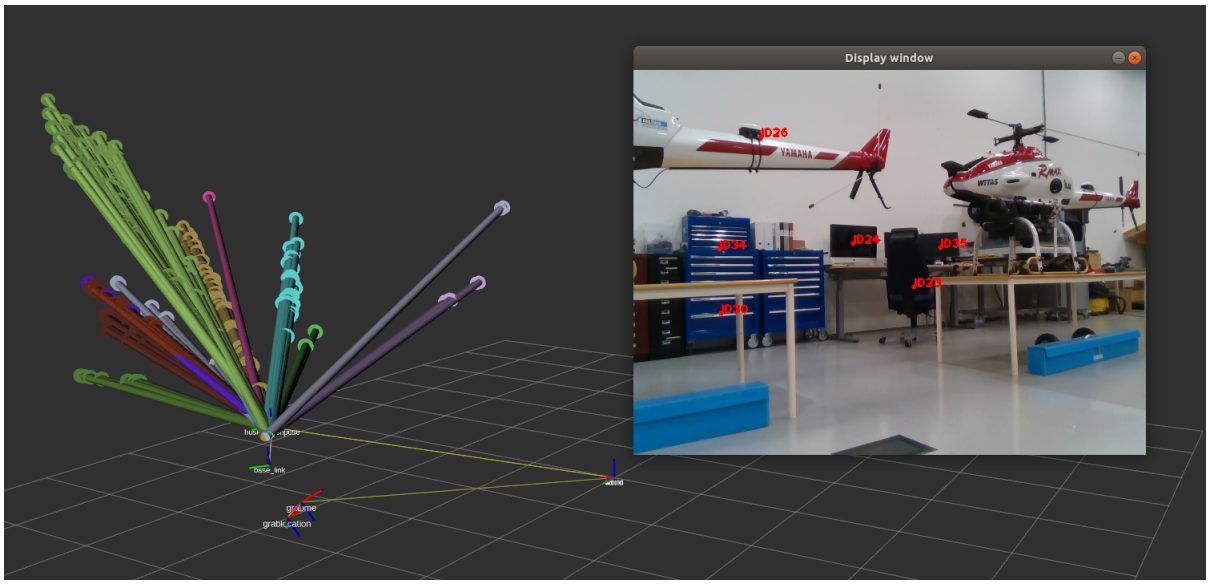


Figure 8.2: Vector Pose Observations

For each object(ID) there is a different arrow with a different colour. The reason for the multiple arrows(the duplicated ones) is because they do not get deleted during the running time. This shows that the tracker is quite accurate because where is supposed to be the object, the arrow's colour does not change but, on the other hand, we can see that the distance for a stationary object situated in a certain position changes a bit. This will use in

the test part for what concerns the stability of the object distance algorithm, and the reliability of object detection algorithms.

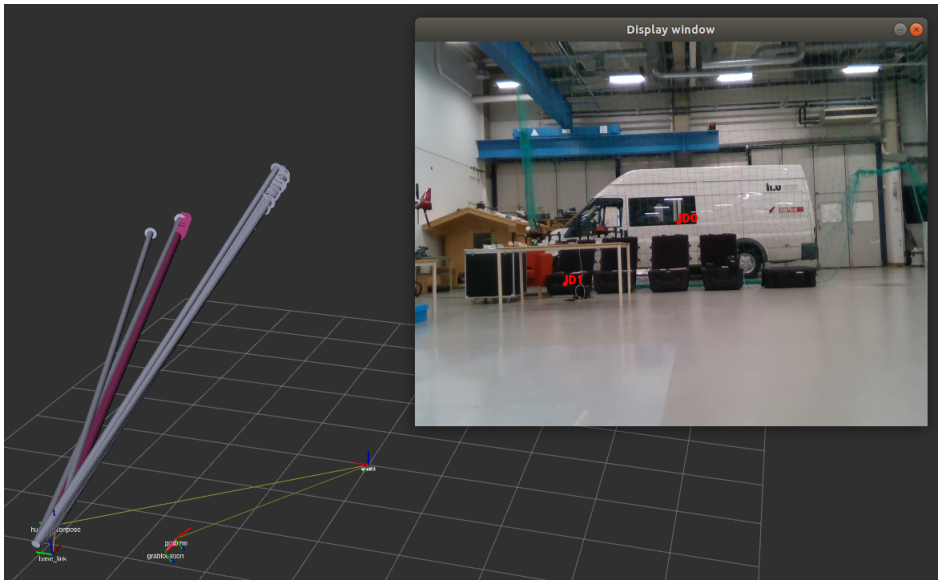


Figure 8.3: Vector Pose Observations

8.2.2 Evaluation

We did not conduct any tests on this part because we build it to use it as the test itself. We will see in Chapter n. 9 the use we have been done of it.

8.3 Feature State Printer

In order to print the map that the EMS(Environment Modelling System) builds with the data it takes from the observations published by the *object_distance* node we built this tool.

8.3.1 Implementation

Using RVIZ tool we built something that we could use to have a graphical representation of the map the robot is building with the help of the observation made and analyzed by the camera. It uses the *RTMappingNodelet/request_features_state* service to take the information it needs. As we can see from the picture(8.4) the sphere represents the landmarks.

The overlap between the sphere and the arrow is due to the arrow aims to the centre of the sphere.

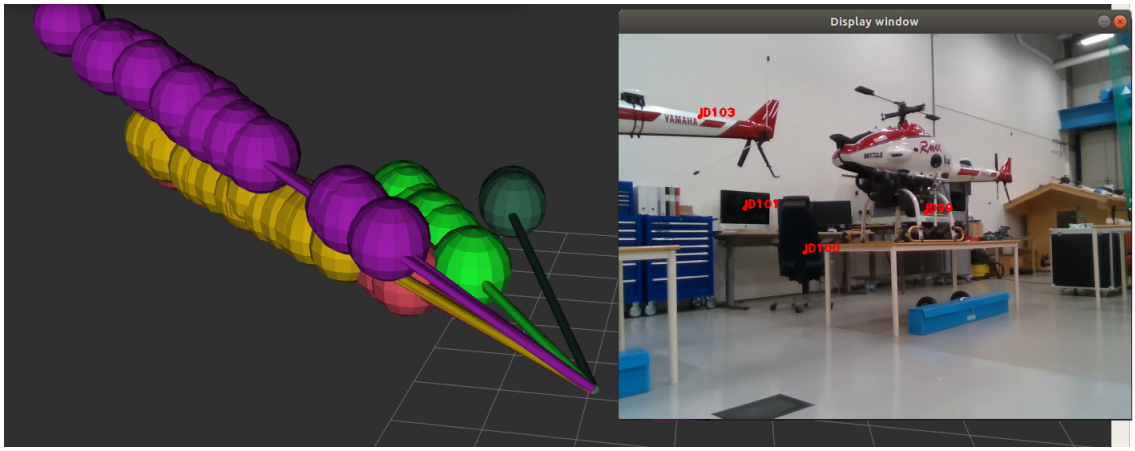


Figure 8.4: Feature state printer

8.3.2 Evaluation

For this tool, as in the previous point, we did not conduct any tests on it because we build it to use it as the test itself. We will see in Chapter n. 9 the use we have been done of it.

Chapter 9

Test and Results

In this chapter, we describe the test part of this work, in order to evaluate it and to test the strengths and weaknesses. This is a small introduction to the tests done. We will analyze and discuss them in the next chapters.

- Test1: Label accuracy.

  To perform this test, some frames are randomly taken and checked. It is analyzed how many possible objects are actually bounded from the detection algorithms.

- Test2: Reliability of detection. Does the label fit the object?

  To test the reliability of the detection a series of random frames are analyzed. We took a certain number of frames and we checked in how many of them the objects inside are correctly detected(ratio object/correct label).

- Test3: Tracker.

  To test the performance of the tracker a series of random frames are analyzed. We calculate out of "x" number of frames how many times the objects are correctly tracked in them.

- Test4: Object distance.

  To test the correctness of the object distance computed from the software we make a comparison between the one from it and a measure made without. To take these measurements we used the IPS(Indoor Positioning System) provided inside the lab. The system gives as output the coordinates of the object, that compared with the position of the

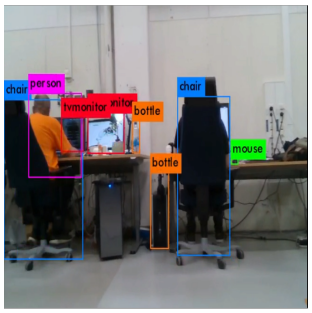
robot and with a simple calculation of the distance between two points, gave us the real distance between the object and the robot.

9.1 Test 1: Label accuracy

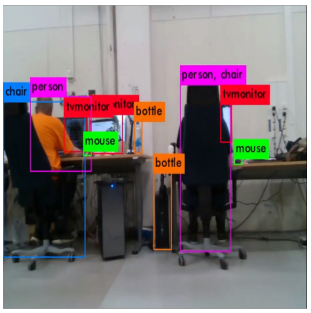
Here we analyze the accuracy of the label. Are all the possible labelled object actually classified inside the frame? As we can see from the Figure 9.3 most of the bounding boxes are correct around the object except for the fact that not all the possible bounded objects are really bounded.

On the left Yolo, on the right ResNet. As we can see from these pictures both the algorithms work fine. ResNet seems to work better, in fact, it recognises the helicopter in the front that YOLO does not seem to detect. Also, the small case on the bottom right is recognized in 3/4 of the frames from ResNet while YOLO just 1/4.

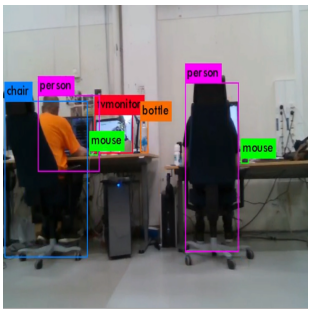
The pictures show how the ResNet algorithm seems to detect (bound) more objects than YOLO. But we will analyze later the reliability of these detections.



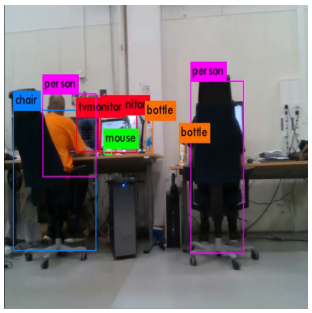
(a) frame at time *t*



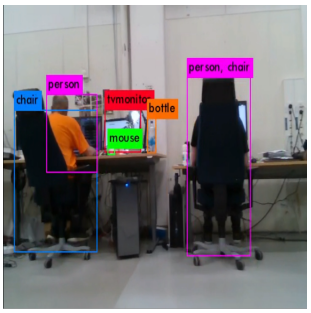
(b) frame at time *t+1*



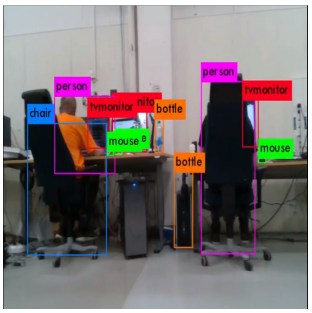
(c) frame at time *t+2*



(d) frame at time *t+3*



(e) frame at time *t+4*



(f) frame at time *t+5*

Figure 9.1: YOLO's frames.

In order to analyze better what it seems to be the worst algorithm, we

take as sample some YOLO's frames. Watching the figure (9.1) we can see the same situation as before. But this time just the "bottle" in the middle-bottom is missed in 3/6 frames. Considering, however, the multitude of objects within the scene. Comparing the two figures(Fig. 9.1 and 9.3) and analyzing the possible causes of the problem, the main differences come out from the dimension of the object and from the quality of the image. The dimension is also based on the camera distance from the object. On the moving of the robot, when the distance decreases, we can see that the detection is more precise. Based on empirical measurements an optimal distance (camera-object) should be less than 2 meters.

Those problems affect both algorithms but YOLO seems to suffer more than ResNet of it.
The same results can be seen in details in the figure 9.2. This graph is taken from [22].

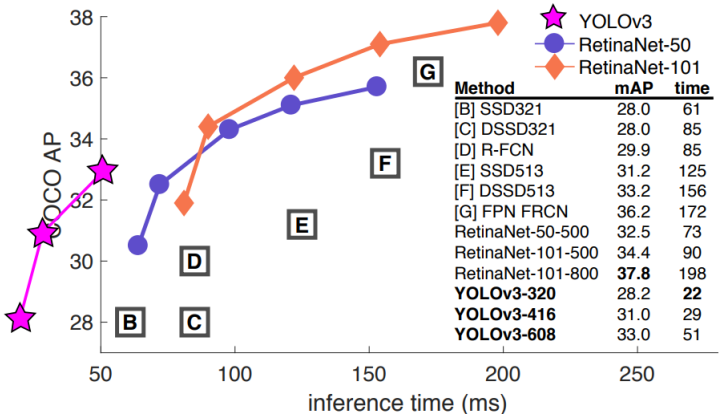


Figure 9.2: YOLOv3 compare performances.

9.2 Test 2: Reliability of detections

We already faced the problem of not labelling from the detection algorithm. Here we are asking: Does the label fit the object? The label just released is reliable? Taking as sample the figure 9.3, where we have both algorithms compared, we can see that the object labelling is quite different from object to object. Before starting the analysis of the single labels we want to remind that each detection algorithm publish inside the message also a percentage of reliability of the pair object-class detected. On the ResNet is published in the

corner after the name of the class, for YOLO is just published inside the ROS message. So, a first decision, concerning the veracity of the label provided by the object detection algorithm should be inserting a certain "trust" threshold beyond which we can accept that specific label. On an empirical basis, we can say that a fair threshold is 60%. For example we can see from the images that the "chair" at the bottom right has a percentage equal to 40%, like the "bench" in the left part. While if we take the two helicopters, we can see how the percentage is clearly higher. As we can see from these examples the first prediction is wrong(<60%) while the second one is right(>60%).

We previously examine some aspects that are strictly related to this analysis. The distance from the object is one of them. We already said that the distance influences or not the labelling of the object and another consequence is the reliability of that label. Less distance leads to less errors. Another aspect that can influence the labelling process is also the light condition due to the environment. In a condition of too much shadow, the process fails. So, in good environmental condition, we can be satisfied with the detection made from both algorithms.

Comparing YOLO(Fig. 9.1) and ResNet(Fig. 9.4) we can see how ResNet seems to be more reliable for labelling instead of YOLO.

Another aspect that we want to analyze regarding the veracity of the labels concern the computational speeds of the two algorithms. The times in which the individual frames of the YOLO and ResNet images were taken in order to make a visible comparison have been placed in the table 9.1.

	ResNet	YOLO
t	57:92	53:03
t+1	58:04	53:06
t+2	58:11	53:10
t+3	58:18	53:13
t+4	58:25	53:18
t+5	58:37	53:21

Table 9.1: ResNet vs YOLO captures.

We can see how YOLO gives result quicker then ResNet. Probably this fact gives to ResNet more accuracy then YOLO. So, even if we have an optimal distance and a good image quality, an important aspect is the computational complexity of the algorithm for the robot. Also because the robot has to handle all the sensor it is employing at the same time.

Also these results can be seen in the figure 9.2.

9.3 Test 3: Tracker

We already explained in chapter n.6 the two techniques used to build the tracker and in chapter n.8 how we combined them. In the first version of the algorithm, there was just the first technique (SURF+FLANN) and the tracker was really inaccurate compared to this new one. In the figure 9.5 we can see some frames taken every 0.5 seconds one from each other.

The picture is quite small but we can clearly recognize the IDs of the 4 main objects.

- ID 22 : Helicopter;
- ID 23 : Chair;
- ID 24 : TV;
- ID 26 : Helicopter;

All the IDs does not change between frames, to show that the tracker does a good job. Calculating the accuracy(ratio correct object tracked/frames) over almost 260 frames(more than 1 minute of working) we got 91%, with an error equal to 9,84%. The main failure aspect of the tracker is when the detection object algorithm gives to an object a wrong label(different from the previous frame). This leads the tracker to an error because the tracker is optimized to work with objects having the same label.

In the figure 9.6 we can see the propagation error from the object detector to the tracker. We can clearly see how the label changing from person to chair, changing also the ID given as output.

9.4 Test 4: Object distance

In figure 9.7 is showed the full software suite working. On the left, we can see the combination of what the robot actually sees (represented as a cloud) and the distance calculated by our algorithm (arrow between robot and cloud). The starting point of the arrows in the ground robot where we can observe an example of the ROS TF system (chapter 4). On the right of the figure,

the tracker. The objective of this test is precisely to compare the measurement obtained by our system with the ones obtained through the Indoor Positioning System present in the laboratory.

In figure 9.8 we can see how the test was conducted. The robot's position was extracted by subscribing to the "/husky/viconpose" channel. The following tables (tab. 9.2 and tab. 9.3) show the points used for measuring distances. While regarding the distances obtained by our algorithm, they have been empirically extracted as reported by the same figure.

As we can see from the tables there is a fair difference between the measure calculated by the camera and the ones calculated using the IPS. This error seems to increases as the distance of the object increases from the robot.

Object detected	Object location	Robot location	Distance with the camera	Distance with IPS
Helicopter	x = 5.979 y = -1.859 z = 0.896	x = 2.990 y = 1.766 z = 0.533	3.455 m	4.712 m
Truck	x = 2.271 y = -7.756 z = 0.108	x = 2.664 y = 4.302 z = 0.517	7.778 m	12.071 m
Truck 2	x = 2.271 y = -7.756 z = 0.108	x = 2.697 y = 3.503 z = 0.523	7.925 m	11.275 m
Box	x = 4.801 y = -5.496 z = 0.960	x = 2.697 y = 3.503 z = 0.523	7.867 m	9.252 m
TV	x = -5.864 y = -2.971 z = 1.520	x = 2.588 y = 1.031 z = 0.543	6.651 m	9.402 m

Table 9.2: Table test 4.1.

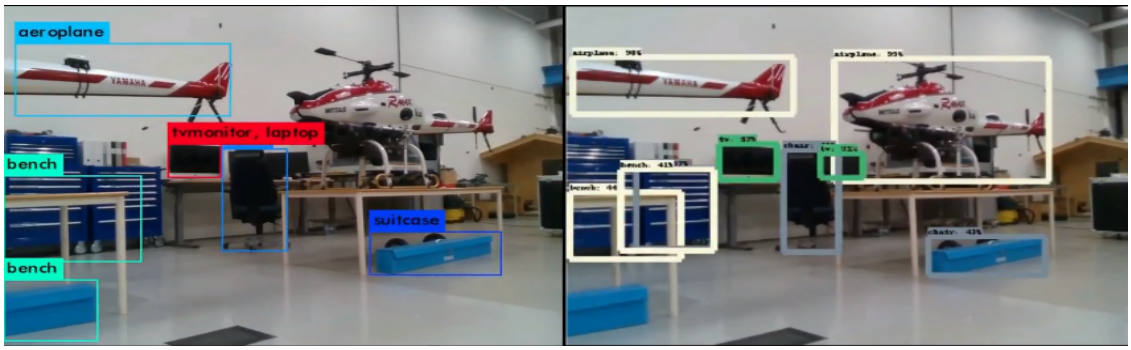
From the data obtained it would also seem that the measurement, once exceeded a certain threshold, is no longer very effective. This could be due to the power limitation of the camera mounted on the robot. Let's remember to be the Intel® RealSense ™ depth camera D435 with a range that is actually around 10m. In fact, the measures that are around or exceed this threshold are the most incorrect. Other problems could be related to scene and lighting

Object detected	Object location	Robot location	Distance with the camera	Distance with IPS
Helicopter	x = 5.979 y = -1.859 z = 0.896	x = 1.488 y = -0.607 z = 0.560	5.759 m	4.674 m
Truck	x = 2.271 y = -7.756 z = 0.108	x = 2.680 y = 3.745 z = 0.523	8.015 m	11.516 m
Truck 2	x = 2.271 y = -7.756 z = 0.108	x = 2.673 y = 3.805 z = 0.520	7.485 m	11.575 m
Box	x = 4.801 y = -5.496 z = 0.960	x = 2.673 y = 3.805 z = 0.520	8.288 m	9.551 m
TV	x = -5.864 y = -2.971 z = 1.520	x = 3.127 y = 1.467 z = 0.536	6.756 m	10.075 m

Table 9.3: Table test 4.2.

condition. Unfortunately, there is not much we can do to correct this mistake. It would be useful do deeper investigation on it, understanding if the error is related to the instrument or to the data processing. An other thing perhaps possible, would be to rely not only on the camera to measure the distance from an object but also on other sensors such as a laser pointer or similar. However it would lead to other problems due to the measurement of the distance by the same.

(a) frame at time t

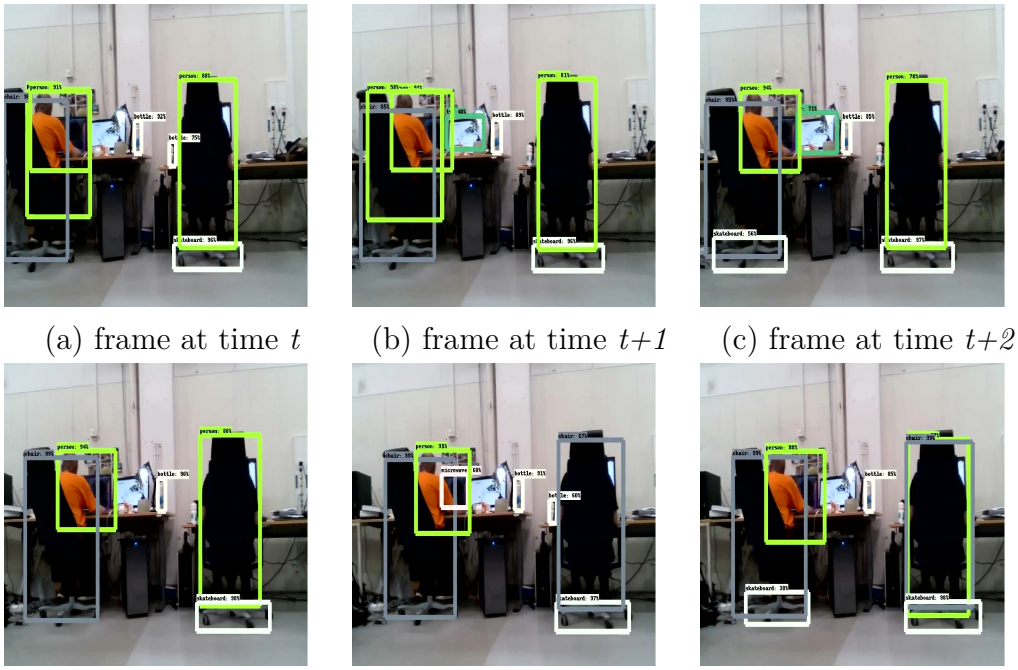


(b) frame at time $t+1$

(c) frame at time $t+2$

(d) frame at time $t+3$

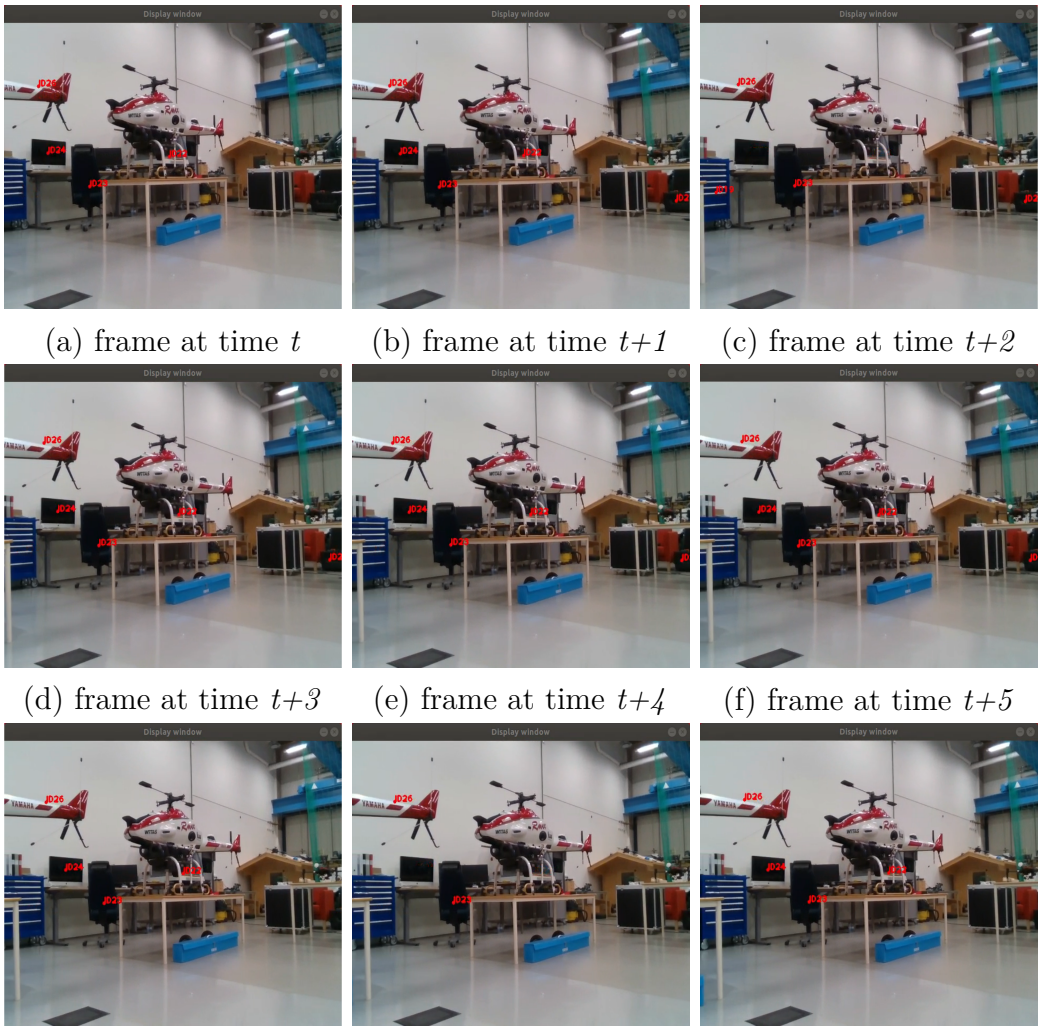
Figure 9.3: Sample comparison between YOLO (left) and ResNet (right) detection algorithms.



(a) frame at time *t* (b) frame at time *t+1* (c) frame at time *t+2*

(d) frame at time *t+3* (e) frame at time *t+4* (f) frame at time *t+5*

Figure 9.4: ResNet's frames.



(a) frame at time *t* (b) frame at time *t+1* (c) frame at time *t+2*

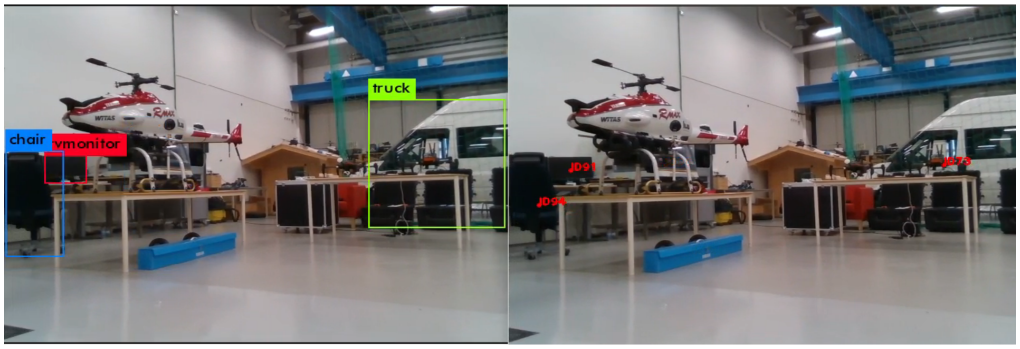
(d) frame at time *t+3* (e) frame at time *t+4* (f) frame at time *t+5*

(g) frame at time *t+6* (h) frame at time *t+7* (i) frame at time *t+8*

Figure 9.5: Tracker.

(a) frame at time t

(b) frame at time $t+1$



(c) frame at time $t+2$

Figure 9.6: Tracker failure.

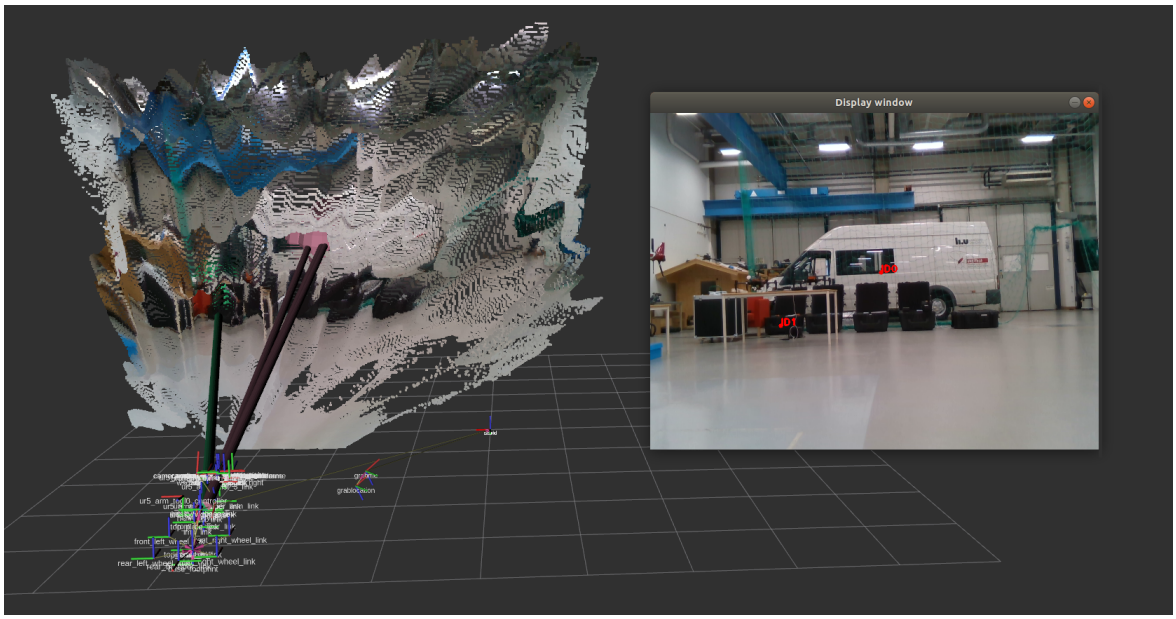


Figure 9.7: RVIZ environment. Tracker on the right and what the robot effectively see with the distance on the left.

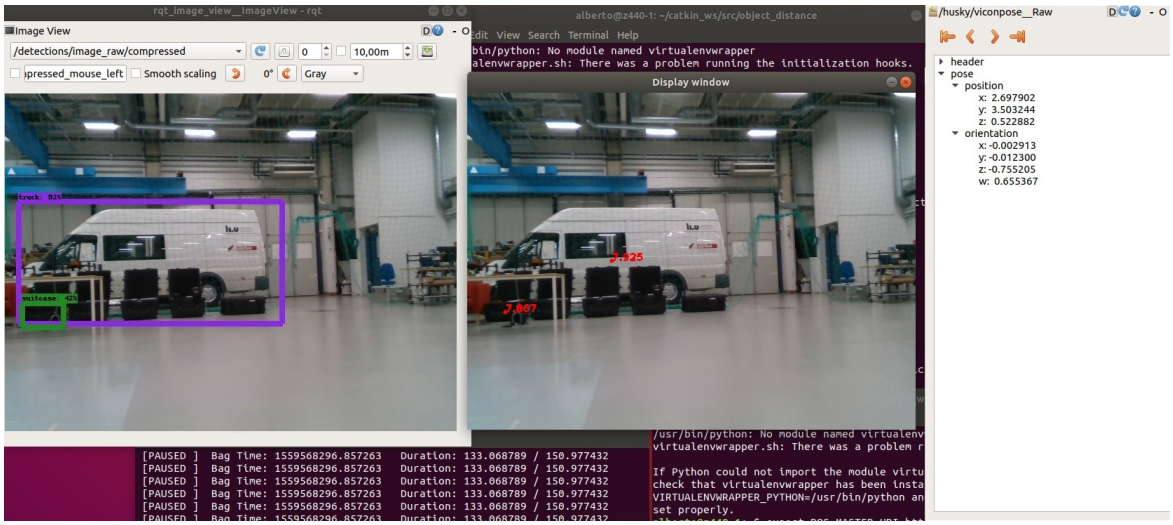


Figure 9.8: Test n. 4. On the left, the output from ResNet detection algorithm, in the centre the object distance from our algorithm and on the left the position of the robot based on the IPS measure.

Chapter 10

Conclusions and Future works

In the introduction, we presented the SLAM problem, in short, a process that constructs a map of an unknown environment by an unmanned vehicle where it can keep track of itself.

The purpose of this thesis was to offer a solution to this problem. To do it, we implemented a deep-learning algorithm for object detection. This should provide us a tool to identify the objects inside the environment and tag them as moving or not moving. The work we have done gives us a connection between the object detection algorithm and the SLAM; it also allows us to track the objects between frames. All of this in order to improve the accuracy of the map and to help the robot to track itself inside. As we can see from the last chapter (chapter n. 9) we also conducted some tests in order to evaluate the work. We started from the algorithms. From the tests conducted we did not really establish a best one. We have shown some strengths and weaknesses of each one. This can help to choose one instead of the other, based on its use. If we aim to have more accuracy then speed we should choose ResNet. Otherwise a faster algorithm but less accurate, YOLO (as reported in chapter 9.1 - figure 9.2). We conducted other tests on the tracker, to prove its robustness. The results there, are quite good. The accuracy (computed as the ratio correct object tracked/frames) is around the 91%, with an error equal to the 9,84%. The implementation of the tracker is fundamental because it can be used, to help the object detection algorithms, to decide which object is a moving or static one. The last test we have conducted is related to the measure of the distance with the support of the depth camera. The results show, in a range of 10m (the measure given by the

producer), an error approximately around a meter. But, on the other hand, the correlation between the colour and black and white cameras(chapter n. 8.1.1) give us more data in order to be more accurate for the measure of each object by our system.
Considering the result obtained from the tests, we can be generally satisfied. The aim of improving the perception of the environment by the use of an object detection algorithm it is been reached. In fact, we were able to answer most of the research questions defined at the early stage of the work.
Although we did not identify the best object detection algorithm, we made a comparison between YOLO and Resnet; this feedback can be used in the future in order to choose the most adequate one, based on the field application and the study case. In the analysis conducted the results confirm our expectations.
An other important result is obtained in the tracker system. In fact we can consider the accuracy reached, about 91%, a good result. This allowed an improvement of the map reliability, on the basis of the observation made. We are now able to recognize the same object in different frames, enhancing in this way the map building.
An outcome we did not expect is the inaccuracy of the distance value determined with the depth camera. Nevertheless, the chosen method was useful to the full system, because of the reason explained before. It would be recommended to carry on deeper investigation on this matter, establishing if the error obtained can be attributed to the instrument mounted on the robot or to the unsatisfactory data processing, that can be improved.

Unfortunately, we did not have enough time to face all the objectives settled at the beginning of the thesis work. In the actual situation all the objects we identify are sent to the EMS as observations and considered as non-moving objects; so, viewing all of them as possible landmarks. A further improving step of the research, for the future use of this algorithm, would be classify and study those observations, in order to use or discard them to increase even more the accuracy of the map. We would like to obtain best results from our algorithm, starting by the use of the test's results carried out during the study. Our plan is to increase as much as possible the accuracy of the tracker, in order to use it for the classification of the objects just discussed.

Acronyms

AI Artificial Intelligence. ii

API Application Programming Interface. 42

BSD Berkeley Software Distribution. 27

CNN Convolutional Neural Network. 5

DCT Discrete Cosine Transform. 40

EMS Environment Modelling Simulator. 66

Fast R-CNN Fast Region-based Convolutional Network. 14

Faster R-CNN Faster Region-based Convolutional Network. 14

FLANN Fast Library for Approximate Nearest Neighbors. 36

GPS Global Positioning System. 1

GPU Graphics Processing Unit. 12

IMU Inertial Measurement Unit. 42

ML Machine Learning. 4

MLP Multi-Layer Perceptron. 8

NN Neural Network. iv, 5

PCC Peaks of Cross-Correlation. 40

PHash Perceptual Hash. v, 39

R-CNN Region-based Convolutional Network. 13

R-FCN Region-based Fully Convolutional Network. 15

ReLU Rectified Linear Unit. 10

ResNet Residual Network. 22

RoI Region of Interests. 14

ROS Robot Operating System. 26

RVIZ Ros Visualization. 27

SIFT Scale-Invariant Feature Transform. 35

SLAM Simultaneous Localisation And Mapping. ii

SURF Speeded-Up Robust Features. 36

TF Transform Library. 27

YOLO You Only Look Once. 18

Bibliography

[1] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955". In: *AI magazine* 27.4 (2006), pp. 12–12.

[2] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[3] Jesse Alpert & Nissan Hajaj. *We knew the web was big...* 2008.

[4] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[7] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Region-based convolutional networks for accurate object detection and segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), pp. 142–158.

[9] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[13] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[14] Hugh F Durrant-Whyte. "Uncertain geometry in robotics". In: *IEEE Journal on Robotics and Automation* 4.1 (1988), pp. 23–31.

[15] Randall C Smith and Peter Cheeseman. "On the representation and estimation of spatial uncertainty". In: *The international journal of Robotics Research* 5.4 (1986), pp. 56–68.

[16] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. "A solution to the simultaneous localization and map building (SLAM) problem". In: *IEEE Transactions on robotics and automation* 17.3 (2001), pp. 229–241.

[17] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. "Simultaneous localization and mapping with sparse extended information filters". In: *The international journal of robotics research* 23.7-8 (2004), pp. 693–716.

[18] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[19] Marius Muja and David G Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration." In: *VISAPP (1)* 2.331-340 (2009), p. 2.

[20] Marko Bjelonic. *YOLO ROS: Real-Time Object Detection for ROS*. `https://github.com/leggedrobotics/darknet_ros`. 2016–2018.

[21] Karol Majek. *Tensorflow Object Detection API in a ROS node*. `https://github.com/karolmajek/object_detection_tensorflow`. 2018.

[22] Joseph Redmon Ali Farhadi and J Redmon. "YOLOv3: An incremental improvement". In: *arXiv preprint arXiv:1803.10827* (2018).