Politecnico di Torino

Master degree in Mechatronic Engineering

in collaboration with Reply spa

"Deep learning techniques for programming a collaborative robotics system"

Supervisor:

A. Rizzo

Supervisors at Reply:

A. Tessuti

A. Grosso

Candidate:

Nicoletta Speraddio (s. 264820)

a.a. 2020/2021

SUMMARY

1. Introduction

The goal of this project is to obtain a robotic system able to accomplish collaborative tasks. Traditional robotics does not fight with the same problems of collaborative one. We have to distinguish traditional robots and collaborative robots. The first ones have specific characteristics and structures, because of human interaction is not present. User controls the robot outside its operating range. An example of traditional robot is the industrial arms shown in the figure 1.1:

Figure 1.1

Its constraints are relative to the task it performs. It is important to note that it must not come into contact with users. It works autonomously without any interactions with human. This is a fundamental aspect that influence the control methodology adopted for.

As robots support increases during last years, also the kind of tasks robots perform are much more different and various. Depending on the field of interest, robots can work singularly and insulated from users working zone or can be used in collaborative functions. These last ones imply different constraint and structure because of safeness in the working area.

An example of collaborative robots is shown in figure 1.2:

Figure.1.2

The idea for the thesis project is to obtain a robotic arm (didactic size) suitable for expert and non-expert users. The challenge is to program basic movements without needing programming skills. 'Programming by Demonstration' (PbD) is one of the most used techniques to solve this kind of problem. PbD has become a central topic of robotics that spans across general research areas such as human-robot interaction, machine learning, machine vision and motor control. This technique has grown importantly during the past decade. The rationale for moving from traditionally programmed robots to robots programmed via user interaction is threefold. [1]

Collaborative robotics requires some components (hardware and software) that must be integrated together with the robot in the overall system. The core of the system is the robotic arm, to which peripheral devices can provide the necessary functionality for the collaborative purpose. For this purpose, it is necessary to set up a capture environment in which the movement to be communicated to the robot will be shown. Stereo vision is the basis of this application, because the movement must be perceived in three dimensions. The acquisition is done from two different points of view. This double vision allows to reconstruct a 3D point/space/object, based on the methodology known in stereo vision theory, explained in a simplified way in the following figure 1.3:

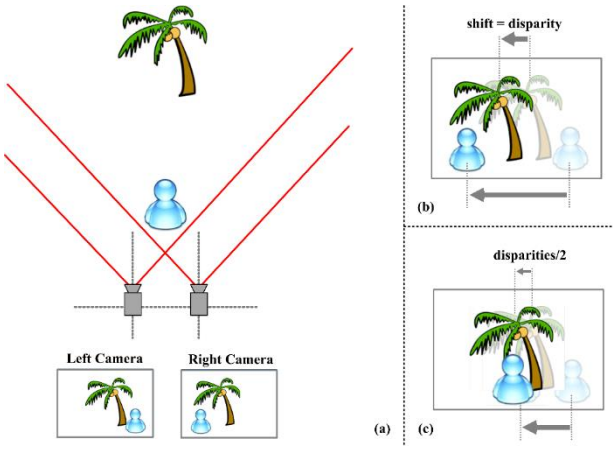


Figure 1.3

After reconstruction of the demonstration space, the parameters obtained in solution are communicated to the robot. In combination with stereoscopic acquisition, deep learning is used to detect the object of interest. For this specific case, the demonstration is performed by the user, who shows the poses by movement of his right arm. The support played by the use of deep learning techniques is needed first to detect the key points of the user's arm and then to correctly map those points. These points are used to formulate the model corresponding to the human arm, making sure to create analogies with the specific joint mechanism of the robotic arm. Point recognition is possible through the use of an open-source library (OpenPose), based on two neural networks trained on human image databases. This library is recognized as the first open-source system used for real-time 2D key point detection; it is used to process frames containing one or more human subjects (multi-detection). For this specific case, the key points detected are the neck, shoulder, elbow and wrist (of the right arm). The performance of OpenPose has some hardware requirements in order to achieve the best performance results. To verify that the performance was purely related to some of these requirements, the required architecture was simulated using Amazon AWS services. Then, a web service was implemented in order to make the processing independent of the instance from which it was invoked.
In the next chapters, each aspect will be analyzed in more detail. The software and hardware supports have been chosen following the analysis of the results obtained during the experimental phases. The applications of industrial type that could be based on this system are various, being the development not finalized to the field of action but to the functionality of the system as collaborative. In general, the idea of the project was born to deal with the problems

of collaborative robotics; this field breaks away from traditional robotics based on the use of robots isolated from the user, and opens up to robot-human interaction. Collaborative robots are rapidly providing important improvements in productivity, safety, ease of programming, portability and cost, making them usable for many new applications. The contact between the human and robot worlds implies the evaluation of different approaches and requirements compared to traditional robotics. In order to have control over the interaction, it will be necessary to provide the robot with perception, as to follow safety protocols while performing actions of interaction with the user. The first chapter introduces collaborative robotics in terms of evaluations and requirements, necessary to define the main characteristics of a collaborative workflow.

2. Collaborative Robotics

Robots have existed in industry for more than half a century. Initially, they were placed in steel cages or in environments that ensured safety for users in the workspace. The first applications in which they were used were the use of large-scale instrumentation, as well as the assembly of components for large structures (e.g. cars). The manufacturing horizon of Industry 4.0 encompasses a shift away from automated production towards a concept of intelligent manufacturing. The unique feature of Industry 4.0 is to meet customer demand for variations in products in real time in a small batch size. This allows a manufacturing system to meet individual customer needs without affecting the production time for setting up and reconfiguring an assembly line. For a smart robotic factory to function in the context of Industry 4.0, high productivity and flexibility become the priority for a manufacturing environment. To address this issue, robots will be employed in most manufacturing processes; although the robot figure becomes predominant, the human worker must remain in the work area to accomplish the remaining tasks for which robots cannot be trained or to supervise automated work cycles. The constant human presence in or near the range of the intelligent robot leads to an increasing focus on the safety aspect. The conventional approach is to expose human workers to a safeguard distance from the robot, this kind of limitation of the robot's range of action is achieved by implementing control techniques on the robot's movement.[3]

2.1 Physical Human-Robot Interaction (pHRI)

Imagine a robot performing a manipulation task next to a user, such as moving the user's coffee cup from a piece of furniture to the table. As the robot moves, the person might notice that the robot is carrying the cup too high above the table. Knowing that the cup would break if it were to slip and fall from so high, the person easily intervenes and starts pushing the robot's end-effector down to bring the cup closer to the table. In these cases, the moment the human lets go of the robot, it resumes its original behavior - our robot would bring the cup back up too high, requiring the person to continue to intervene until it has finished the current action. Although such control strategies ensure a quick reaction to unexpected forces, returning the robot to its original movement from a fundamental limitation of traditional pHRI strategies: robots do not take into account that human interventions are often intentional and occur because something is being done wrong. While the robot's original behavior may have been optimal with respect to the robot's predefined objective function, the fact that human intervention was required implies that this objective function was not entirely correct. Returning to our example, if the person is applying forces to push the robot's end-effector closer to the table, then the robot should change its objective function to reflect this preference, and complete the rest of the current task accordingly, keeping the cup lower. Ultimately, human interactions should not be thought of as disturbances, which disrupt the robot from its desired behavior, but rather as corrections, which teach the robot its desired behavior. These techniques, used in robotic programming, aim to use the pHRI to correct or change the objective function of the robot while the robot is performing its current task. Demonstrations given by the user to the robot, to communicate any corrections or deviations from its initial objective function, can be performed

during the performance of the task or as offline demonstrations, in which the robot changes its function as a result of multiple demonstrations.

2.2 Control strategies for online reactions to pHRI

A variety of control strategies have been developed to ensure safe and responsive pHRI. They largely fall into three categories: impedance control, collision management, and shared manipulation control. Impedance control relates deviations from the robot's planned trajectory to interaction pairs. The robot renders virtual stiffness, damping, and/or inertia, allowing the person to push the robot away from its desired trajectory, but the robot always returns to its original trajectory after the interaction ends. Collision handling methods include stopping, switching to gravity compensation, or re-timing the planned trajectory if a collision is detected. Finally, shared manipulation refers to the assignment of roles in situations where the human and robot are collaborating. These control strategies for pHRI work in real time and allow the robot to safely adapt to human actions; however, through this approach, the robot fails to take advantage of these interventions to update its understanding of the task - left alone, the robot would continue to perform the task in the same way it had planned prior to any human interaction. In contrast, we focus on allowing robots to adjust the way they perform the current task in real time.

- Offline learning of robots' objective functions.

Inverse Reinforcement Learning (IRL) methods explicitly focus on inferring an unknown objective function, but do so offline, after passively observing expert trajectory demonstrations. These approaches can handle noisy demonstrations, which become observations of the true objective. In the developed design, offline learning is based on communicating with the robot about the configuration it should take. It does not involve training via Inverse Reinforcement Learning, but only a single observation and subsequent mirror response.

- Online Human Objective Learning.

While IRL can learn the robot's goal function after one or more demonstrations of a task, online inference is possible when the goal is simply to reach an objective state, and the robot moves in free space. We build on this work by considering general objective parameters; this requires a more complex (non-analytic and difficult to compute) observation model, along with additional approximations to achieve online performance. During project development, this approach is taken to perform collaborative tasks. This involves recognizing the user's hand to reach this point and exchange some objects with him.

a. Safety in pHRI

Safety during interaction with unstructured and dynamic environments is now an established requirement for complex robotic systems. A wide variety of approaches focus on introducing safety assessment methods in order to define a consequent safety-oriented control strategy that

can reactively prevent collisions between the robot and potential obstacles, including a human being. Safety in human-robot interaction (HRI) has gained increasing relevance in industrial environments, where in the near future humans and robots are expected to safely coexist and cooperate, sharing the same workspace. Clearly, this aspect of safety is closely related to the crucial task of collision avoidance. Possible collisions in fact can occur between a robot and a human being, between the robot and potential obstacles, but also with the robot structure itself. The concept of safety field is defined in the range of action of the robot, by means of an evaluation that meets certain requirements. In particular, the safety field concept is:

a. dependent on the relative position and relative velocity between a moving rigid body "source of danger" and a generic moving point in space;
b. dependent on the actual shape and dimensions of the rigid body;
c. efficiently calculated in closed form, allowing real-time applications. A basic definition of the safety field is based on the consideration of a generic body as a source of danger, moving in R3.

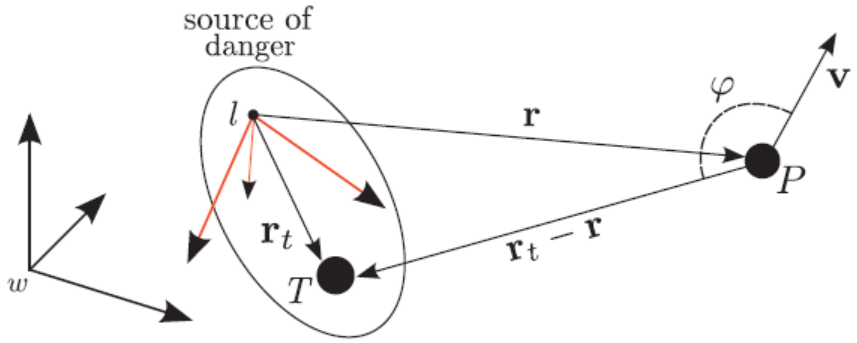


Figure 2.1

A local reference frame is considered such that the position of one of its points T is given by:

$r_t = (x_t\ y_t\ z_t)^T$

The velocity is zero in the introduced frame. r and v denote the position and velocity of a generic moving point in the space P, respectively, expressed in local coordinates. We additionally define the angle between r_t-r and v as:

$$\varphi =< (r_t - r,\ \ v) \in [-\pi, \pi)$$

The Kineto-static Safety Field (KSSF) induced by the motion of a moving point mass on a moving point in space, as the following scalar function:

$$SF(r_t, r, v) = \|r_t - r\|^2(\gamma - (r_t - r)^T v)$$

Where γ is a positive constant such that $SF(r_t, r, v) \in \mathrm{R}^+$. A suitable choice of γ satifies:

$$\gamma \geq \|r_t - r\|\|v\|$$

Nevertheless, note that for $\gamma >> |r_t - r|$ |v| the effort of $(r_t - r)^T v$ becomes negligible. Setting the convenience ρ = $||r_t - r||$ and v = |v|, KSSF satisfies the following conditions:

i. $\frac{\partial SF}{\partial \rho_t^2}(r_t, r, v) \equiv \eta > 0, \ \forall v \geq 0, \forall \varphi \in [-\pi, \pi)$;

ii. $\frac{\partial SF}{\partial \rho_t^2}(r_t, r, v) < 0, \ \forall v \geq 0, \forall \varphi \in (-\pi/2, \pi/2)$;

iii. $\frac{\partial SF}{\partial \rho_t^2}(r_t, r, v) > 0, \ \forall v \geq 0, \forall \varphi \in [-\pi, \pi)$.

The safety field is clearly affected by distance, as highlighted in condition (i), but also considers the magnitude of the velocity vector v (relative velocity between the hazard source and the point where the field is calculated) and the declination angle φ.
As condition (ii) implies, the safety field decreases with the magnitude of the velocity vector v if the point of motion r is directed toward the point r_t (i.e., positive scalar product between ($r_t - r$) and v).
Finally, condition (iii) indicates that for a fixed velocity magnitude, the safety decreases with decreasing declination angle φ.
Notice that, these conditions are equivalent to those satisfies by the elementary kineto-static danger field.[4]

2.3 Dynamic tasks classification and scheduling

Despite the progress in research and the availability of robot models suitable for industrial applications, there are still several unresolved issues due to the continuous change of work processes: some of the aspects to which the most attention is drawn are the correct evaluation of economic viability, the definition of an adequate process plan, the assignment of tasks to humans and robots and the intuitive and fast programming of robots. Task classification is used for workload distribution and detailed task planning. The method is based on the assumption that tasks should be assigned, taking advantage of the different capabilities and resources of humans and robots, regardless of the workload balance. Several works have addressed the evaluation of collaborative robotic cells, especially in the automotive industry, providing a comparison between the conventional robotic cell and the human-robot cooperative cell. The optimal task assignment between workers and robots has been studied in some use cases; therefore, great efforts have been devoted to the safety of human-robot collaboration. Considering the production process divided into work tasks, it is evident that some tasks could be more profitably performed by humans or robots alone, others collaboratively. The human and robotic contribution and the choice of collaboration strategy must be decided in advance by the process designer. The work is divided into several tasks and these are assigned to the human or robot following decision steps based on the analysis of available resources, suitability of the task, minimum operation time required to perform the task. The approach is particularly effective when human-robot collaboration is considered under conditions of spatial or temporal separation in the work cell. Especially in small production runs, more than one solution is possible: the task can be performed by both humans and robots. Manual cells have the additional problem of assigning the load to different workers in a balanced way. Very often, each worker can perform each task and the assignment is just a matter of workload. The human and the robot

have different abilities that should be leveraged as much as possible. Also, there is no need to balance the workload between human and robot. If you take, for example, the collaborative industrial robotic system and its associated work environment, you have to make safety assessments in addition to the workload (ISO Technical Standard 15066[6]). Based on this, it is possible to have different types of collaborative work based on the presence, or not, of temporal and/or spatial separation between humans and robots. The main cases of human-robot collaboration have been classified as:

- Safety-rated monitored stop (temporal and spatial separation):

The collaborative robot stops and remains stopped when the operator is in the workspace.

Figure 2.2

- Hand-guiding (temporal separation):

The operator has a guiding device to move the robot in the intended position. When the operator releases the guiding device, a safety-rated monitored stop is issued.

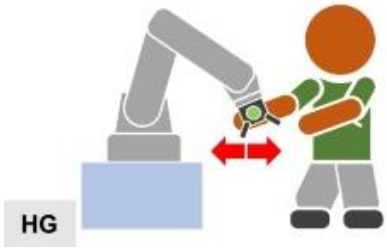


Figure 2.3

- Speed and separation monitoring (spatial separation):

Minimum protective separation distance between the operator and robot system is maintained at all times.

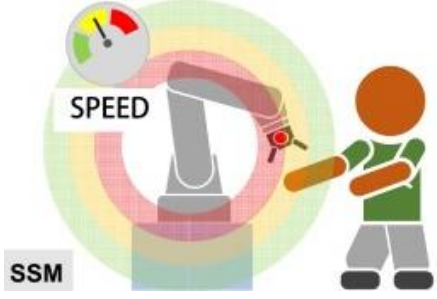


Figure 2.4

- Power and force limiting (workspace sharing):

Physical contact between the robot system (including workpiece) and an operator can occur either intentionally or unintentionally, but exerted forces are limited.

Figure 2.5

In the industrial field it is far more common to find are hand guiding or speed monitoring. They allow the use of standard industrial robots with a minimum refurbishment of the work cell. [5]

2.4 Main applications

Over the past decade, customers are demanding diversified or custom options, and manufacturers are moving toward smaller batches and in some cases made-to-order to meet this demand. Manufacturers are also moving, at the same time, to lower the cost of production such as material logistics on the factory floor. Automation is one answer to these demands, but traditional robotics are not suited for the task. Cobots are the solution to this kind of need. Other factors influencing the current manufacturing environment is that the workforce is aging and younger employees entering the workplace do not see manual-type jobs as attractive as in the past. It has also been reported that traditional robots can only handle about 10% of the new tasks that need to be addressed. End products of automated processes, then with collaborative robotic supports during production flows, with their lower initial cost, quick and easy setup and programming, portability from one application to another, and faster return on investment, are increasingly coming to light for their advantages. Key applications for collaborative robots include packing products into shipping containers, loading and unloading the production line, assembly operations, parts testing, machine maintenance, and workplace material logistics. Problems in adapting robots in the past, such as the difficulty in justifying the investment, worker acceptance of robot safety, programming the robot to perform the desired task, and the difficulty in repurposing robots are all addressed by collaboration. Many of the collaborative robots already on the market can be easily transported by a human from one application to another and quickly programmed for rapid reorganization. Along with the ability to quickly program the application, repurposing is a great way to improve return on investment. Today's collaborative robots are generally smaller and have much less arm inertia than traditional robots, making them much easier to stop quickly and safely. The safety of the collaborative chain depends on the arm stopping almost instantly if it makes contact with a human. Typical collaborative robotic arms can stop in as little as 2 mm of distance. This ensures that no harm occurs to the human or the application. Collaborative robots will be increasingly important in the coming years. It will be a game changer. It may even turn out that collaborative robotics

technology will become the dominant robotics technology in the decades to come, replacing the large one-armed giants that will continue to serve select applications. Clearly, the technology advances the state of the art in robotics and offers many new benefits. The speed at which this change will occur will depend on how both suppliers and their customers approach the technology. Suppliers will need to continue to be effective in educating the market about the benefits and educating on the means to best adapt the technology.[6]

2.5 Case study

Collaborative robots rely on detecting external events with subsequent reaction to them. The way to detect stimuli and produce an intelligent response is to use Machine Learning techniques to support the robot's programming. Machine Learning gives the possibility to make the system intelligent and facilitate the automation of systems based on it. In the developed project, a dual application of the same system is proposed. Starting from the concept of collaborative task, the system lays the foundation on the use of Machine Learning techniques to program the robot. The perception of the surrounding space is fundamental to achieve this goal. The final application is not a discriminating factor for the implementation, it is rather made a point of interest to produce something versatile for different applications. Machine Learning is exploited to recreate a type of robot programming that does not require the development of an algorithm or its understanding by the user who will have to program it. The robot must be made capable of perceiving the pose of a human user. The robot is programmed through imitation of the movements performed by the user. This type of system can be applied in small production, performing basic tasks such as pick-and-place. Considering the goal of making it possible to program the robot without the need to program it traditionally, visual perception becomes necessary. The development of perceptual ability makes it adaptable to a collaborative robotics scenario.

3. Machine Learning

Machine Learning (ML) is an important aspect of business and research in the modern world. It uses algorithms and neural network models to help computer systems progressively improve their performance. Machine Learning algorithms automatically build a mathematical model using collections of sample data and using that data to make decisions without being specifically programmed to make those decisions. [7]

3.1 Deep Learning

Deep Learning, as a branch of Machine Learning, employs algorithms to process data and mimic the thought process, or to develop abstractions. Deep Learning (DL) uses layers of algorithms to process data, understand human actions, and visually recognize objects. Information is passed through several layers, reporting the outputs of the previous layer as input to the next layer. The first layer of a network is called the input layer, while the last is called the output layer. All the layers between the two are called hidden layers. Each layer is typically an algorithm containing one type of activation function. The data analyzed using these techniques are used for the purpose of extracting features that are common to them: feature extraction is another task performed by Deep Learning. There are different types of neural networks in deep learning: convolutional neural networks (CNN), recurrent neural networks (RNN), artificial neural networks (ANN), etc. These types of neural networks are at the heart of the deep learning revolution, and are aimed at supporting modern applications (e.g., unmanned aircraft, self-driving cars, speech recognition, etc.).[8]

3.2 Convolutional Neural Network (CNN)

CNN is probably the most popular deep learning architecture. The recent surge of interest in deep learning is due to the immense popularity and effectiveness of convolutional networks. CNN is now the reference model for any problem based on image processing. In terms of accuracy, they have numerous advantages and high accuracy. The convolutional networks are also successfully employed to systems that need natural language processing. The main advantage of CNN over its predecessors is that it automatically detects important and distinctive features in the analyzed data without any human supervision. For example, given many images of dogs and cats, it learns the distinctive features for each class on its own. CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This allows CNN models to run on any device, making them universally applicable. All CNN models follow a similar architecture, as shown in the figure:

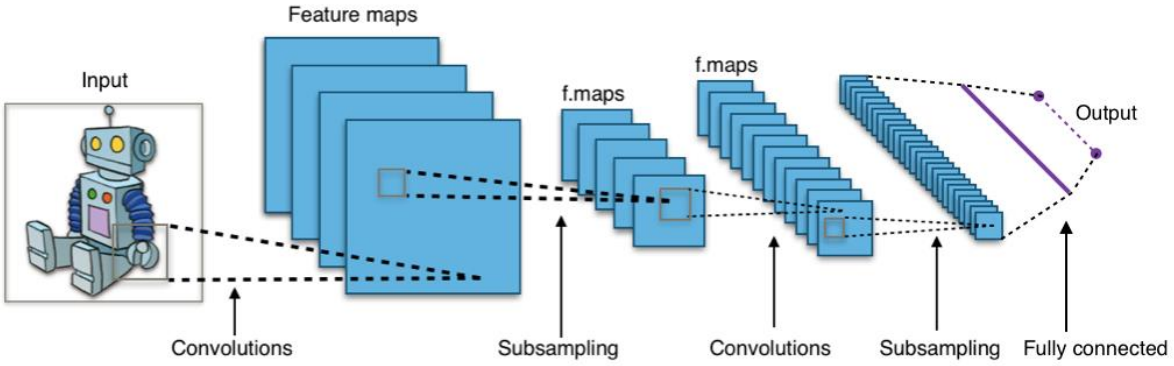


Figure 3.1

There is an input image on which the work must be executed. A series of convolutions and pooling operations are performed, followed by a number of fully connected layers. If multi class classification is performed, the output is softmax. Softmax or normalized exponential function is a generalization of the logistic function to multiple dimensions.

3.2.1 Convolution

The main building block of the CNN is the convolution layer. Convolution is a mathematical operation aimed at joining two sets of information. In our case, convolution is applied on the input data through the use of a convolution filter, with the purpose of mapping the detected features. Below is an analysis of the main components shown in Figure 9: on the left is the input at the convolution level, for example the input image. On the right is the convolution filter, also called kernel, we will use these terms interchangeably. This is called 3x3 convolution because of the shape of the filter (shown in Figure 3.2).

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4		

Feature Map

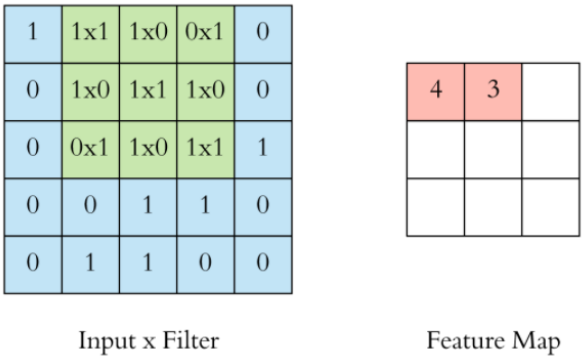


Figure 0.2.2

On the left of the image is the filter: the output of the convolution filter is shown on the feature map. After using the filter to extract the discriminating features, we go to the right and perform the same operation, reporting the result on the feature map. This was an example of a convolution operation using a 3x3 filter. In reality, these convolutions are performed in 3D. The 3-dimensional analysis considers the image as a 3D matrix (height, width and depth dimensions). Depth is expressed in RGB values. A convolution filter has a specific height and width, but during design a filtering that covers the third dimension (depth) is also included. It then becomes a filtering process suitable for analyzing the image in 3 dimensions. Multiple convolutions on an input are performed using a different filter and obtaining a distinct feature map for the additional dimension. The results are then stacked into the detected feature maps and this composition becomes the final output of the convolution process.

Multiple convolutions:

Considering an image 32x32x3 and a filter 5x5x3 (depths are the same for image and filter). When the filter is at a particular location it covers a small volume of the input and the convolution operation is performed as explained. The only difference this time is that the sum of matrix multiply in 3D, not in 2D, but the result is still a scalar. The filter is sliding over the input like above and perform the convolution at each location, aggregating the result in a feature map. The feature map size is 32x32x1 as shown on the right of figure 3.3.

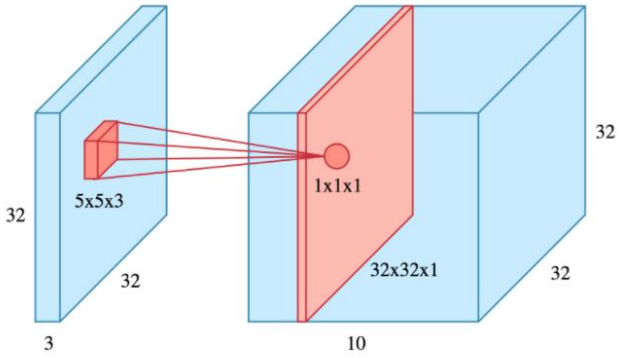


Figure 3.3

If 10 different filters are used the results are 10 different feature maps of the same size (32x32x1). Stacking them along the depth dimension will give us the final output of the

convolution layer: 32x32x10 (as shown in figure 3.2). The height and width of the feature map are unchanged and still 32.

3.2.2 Non-Linearity

For any type of neural network to have high performance, there is a need to consider the aspect of nonlinearity. The values in the final feature maps are not actually sums, they refer to the ReLu function (in the context of artificial neural networks, the rectifier or ReLu is an activation function defined as the positive part of its argument) that is applied as a result of the simple sum. In conclusion, any kind of convolution implies a ReLu operation, without which the network will not reach its true potential.

3.2.3 Stride and Padding

The Stride methodology specifies how much we move the convolution filter at each step. You can have larger steps if you want less overlap between receptive fields. This also reduces the size the resulting feature maps (Figure 3.4), considering the jump in potential positions.

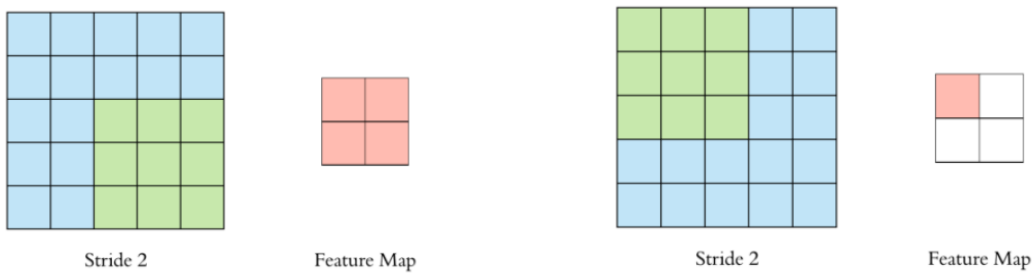


Figure 3.4

The size of the map is smaller than the input because the convolution filter needs to be contained in the input. If a requirement is to maintain the same dimension it is possible to use padding surround the input with zeros (as in figure 3.5):

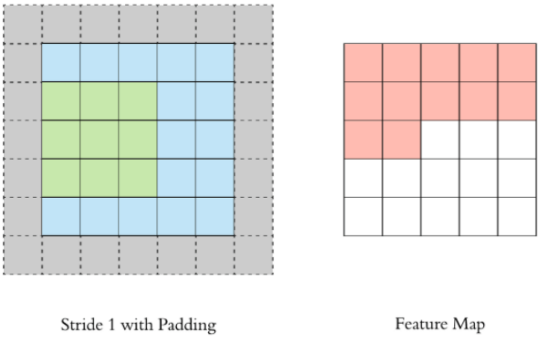


Figure 3.5

The grey area around the input is the padding. Now the dimensionality of the feature map marches the input. Padding is commonly used in CNN to preserve the size of the feature maps,

otherwise they would shrink at each layer, which is typical not desirable. The 3D convolution figures shown in previous figures used padding, this is the reason why the height and width of the feature map is the same as the input (32x32) and only depth changed.

3.2.4 Pooling

After a convolution operation, a pooling operation is performed to reduce the size of the output. This reduces the number of parameters, which shortens the processing time and combats overfitting. In general, there is a trade-off between the size of the space of distinct models that a learner can produce and the risk of overfitting. As the space of models between which the learner can select increases, the risk of overfitting will increase. However, the potential for finding a model that closely fits the true underling distribution will also increase. This can be viewed as one facet of the bias and variance trade-off.[19]
Pooling layers are used to sample each feature map independently, reducing the height and width while keeping the depth intact. Figure 10 shows max pooling using a 2x2 window and stride 2. Each color denotes a different window. Since both the window size and stride is equal to 2, the windows do not overlap.

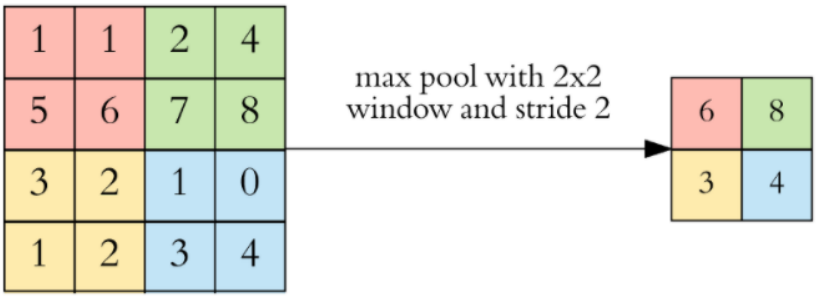


Figure 3.6

Note that with this configuration of windows and stride, the size of the feature map is halved. This is the main use case of pooling, sampling the feature map while retaining important information.
By halving the height and width, we have reduced the number of weights to ¼ of the input. Considering that typically the problem is dealing with millions of weights, in CNN architectures it is critical to reduce this value.
In CNNs, pooling is typically done with 2x2 windows, stride 2 and no padding. Convolution is done with 3x3 windows, stride 1 and padding.

3.2.5 Hyperparameters

Let's consider a convolution layer ignoring pooling. Now 4 important hyperparameters must be decided:

a) Filter size: typically the size is 3x3, but also 5x5 and 7x7 are used depending on the type of application. These filters are 3D and have a depth value of 3, but since the depth of a filter in a given layer is equal to the depth of its input, it is omitted.

b) Number of filters: this is a power of any two between 32 and 1024. Increasing the number of filters results in a more powerful model, but you may run into the risk known as overfitting, caused by increasing the number of parameters.

Usually in the initial layers the number of filters is low and increases progressively as the depth increases.

c) Stride: the default value is 1.

d) Padding: usually used.

3.2.6 Fully connection

After the convolution and pooling layers, a couple of fully connected layers is added to wrap the CNN architecture. The output of both convolution and pooling layers are 3D volumes, while fully connected layers expect 1D vector of numbers. The output of the final pooling layer is flattened, becoming a vector, then the input to the fully connected layer. Flattening is simply arranging 3D volume of number into a 1D vector.

3.2.7 Training

CNN is trained through back propagation with gradient descent. A CNN model can be thought as a combination of two components: the feature extraction part and the classification part. The convolution + pooling layers perform features extraction. For example, given an image, the convolution layer detects features such as two eyes, long ears, four legs, a short tail and so on. The fully connected layers then act as a classifier on top of these features, and assign a probability for the input image being a dog. [9]

3.2.8 Human pose detection using CNN

CNNs recognize spatial features as well as temporal features by comparing pairs of images from two adjacent frames. Traditional feature-based methods, such as those based on color or motion blobs, perform tracking by maintaining a simple model of the target and adapting that model over time. Real-world situations in practice pose enormous challenges to these techniques because: 1. over time, the model of the object may deviate from the original model 2. they lack a discriminating model that distinguishes the object category of interest from others. The main challenge of traditional learning-based and/or tracking-by-detection methods is false

positive matches that lead to the misunderstood association of traces. The reason for these wrong identifications is that these methods rely on applying an appearance pattern or object detector in all possible windows around the target. When the distractor objects are similar to the target, the object detector will generate high and similar detection scores for both the target and the distractors, which will likely cause a drift problem. It is very difficult if we use a head detector because the heads of other people in the crowd can also be seen as exact matches during recognition. The drift problem is alleviated by the use of additional information for detection, this marks the difference between an object detector and a tracker that is able to estimate the position and scale of the target given its previous position and size, as well as the current and previous image frames. CNNs force local feature extraction by restricting the receptive fields of hidden units to be local, based on the fact that images have strong 2D local structures. The features (structures) are learned during offline training. The generality of features learned by testing the CNN tracker can be observed on different scenarios and environments. In fact, tracking by sensing methods can be seen as a special type of learning-based methods, since the feature is learned offline. The usually proposed approach is based on two phases: during the first phase, an appearance model of individuals is built, and during the second phase, people are tracked by detecting these models in each frame. These methods are based on body part analysis or pose estimation. For recognition or detection tasks, primitive feature detectors that are useful on one part of the image are likely to be useful on the whole image. Human identification is the first step for robots to interact with humans.[10] However, identifying a human is a non-trivial process that encounters the problems of highly articulated human body postures and occlusions. CNN combined with the so-called regional proposal becomes the R-CNN that can be applied for object detection. R-CNN shows a great improvement in detection accuracy over conventional feature-based detectors. In R-CNN, the possible objects are extracted from the selective search that proposes 2000 object regions. To accelerate the R-CNN computation, a fast R-CNN is proposed (ROI pooling layer and two full connection layers after CNN layers). The R-CNN improves the original system by joining an ROI proposal layer that gives k-possible region proposals and decides which one contains an object. The depth of the network is of crucial importance, and the main results on the challenging ImageNet dataset all exploit "very deep" models; with an increase in depth, the accuracy may not improve further. One obstacle to improving accuracy is the well-known problem of vanishing/exploding gradients. However, the degradation problem occurs for deeper networks, once the depth of the network increases, the accuracy saturates and then degrades rapidly. Such degradation is not caused by overfitting, adding more layers to an adequately deep model leads to higher training error. The deep learning network is applied for the detection of specific parts of the human body.[11] A development of three different types of body part detector is done using R-CNN (head and shoulder, torso and leg). The method is based on the following contributions:

1. It can handle partial occlusion problems based on the evidence of the visible body parts.
2. It is robust to articulation and viewpoint changes of the observed human object because the constraints of the spatial relation of the parts are flexible, and the final decision is made by multiple part detectors.
3. It uses a simplified fast R-CNN to detect different types of body part.

4. It performs the occlusion compensation via an occlusion map to identify the human object and reject false alarms.

3.3 Role of CNN in the project

OpenPose[2] is the library used during the implementation of the system. It is a Real-time multiple-person detection library, and it's the first time that any library has shown the capability of jointly detecting human body, face, and foot key points.
The pipeline from OpenPose is actually pretty simple and straightforward.
First, an input RGB image is fed as input into a "two-branch multi-stage" CNN. Two branches means that the CNN produces two different outputs. Multi-stage simply means that the network is stacked one on top of the other at every stage. (This step is analogous to simply increasing the depth of the neural network in order to capture more refined outputs towards the latter stages.)
The top branch, shown in beige, predicts the confidence maps of different body parts location such as the right eye, left eye, right elbow and others. The bottom branch, shown in blue, predicts the affinity fields, which represents a degree of association between different body parts.
The term AI now encompasses the whole conceptualization of a machine that is intelligent in terms of both operational and social consequences. With the prediction of the AI market to reach 3 trillion by 2024, both industry and government funding bodies are investing heavily in AI and robotics.[12]

4. Artificial Intelligence in Robotics

Neural networks owe their structure and principle of operation to the structure of the animal nervous system. AI aims to perform tasks related to "perception". Limitations of using deep learning can be distinguished as: 1) Low interpretability of the resulting learned model. 2) Large volumes of training data, resulting in the need for significant computational power Building on advances made in mechatronics, electrical engineering, and computer science, robotics is developing increasingly sophisticated sensorimotor functions. They give machines the ability to adapt to their changing environment. The autonomy of a robot in an environment can be divided into perception, planning, and execution (manipulate, navigate, collaborate). The main idea of the convergence of AI and Robotics is to try to optimize its level of autonomy through learning. Robots with intelligence have been attempted many times. Although the creation of a system that exhibits human-like intelligence remains elusive, robots that can perform specialized autonomous tasks, such as picking up objects and putting them down, driving a vehicle, etc., are still possible. Another important application of AI in robotics is the task of perception. Robots can perceive the environment by means of built-in sensors or systems based on computer vision. Perception is not only important for planning, but also for creating an artificial sense of self-awareness in the robot. This allows supporting the robot's interactions with other entities in the same environment. This discipline is known as social robotics. It covers two domains: human-robot interaction (HCI) and cognitive robotics. The field of HCI involves robotic perception of humans such as understanding activities, emotions, nonverbal communications, and the ability to navigate an environment along with humans. The field of cognitive robotics focuses on providing robots with the autonomous ability to learn and acquire knowledge from sophisticated levels of perception based on imitation and experience. It aims to mimic the human cognitive system. In cognitive robotics, there are also models that incorporate motivation and curiosity to improve the quality and speed of knowledge acquisition through learning. AI has continued to break all records and overcome many challenges that were unthinkable less than a decade ago. In general, a typical robotic assembly operation involves working with two or more objects/parts. Each part is a subset of the assembly. The purpose of the assembly is to compute an order of operations that brings the individual parts together to make a new product. To learn the execution of assembly operations, a robot must first estimate the pose of the parts to be machined, then an assembly sequence can be programmed directly on the robot or "demonstrated" by the robot user. Robotic assembly remains one of the most challenging problems in the field of robotics research, especially in environments not structured to accommodate such. Traditional robots require users to have programming skills, which makes robots out of reach of the public. Today, robotics researchers have been working on a new generation of robots that could learn from demonstration and do not need programming. These new robots could sense human movements using their sensors and mimic the same actions that humans do. Imitation takes place when an agent learns a behavior by observing the execution of that behavior by a teacher. This was the starting point for establishing the characteristics of robot imitation:

i. Adaptation
ii. Efficient communication between the teacher and the learner
iii. Compatibility with other learning algorithms
iv. Efficient learning in an agent society.

Processes in robot imitation have been identified, namely perceiving, understanding, and doing. They can be identified as: observing an action, representing the action, and reproducing the action. This distinction is shown in the following figure 4.1:

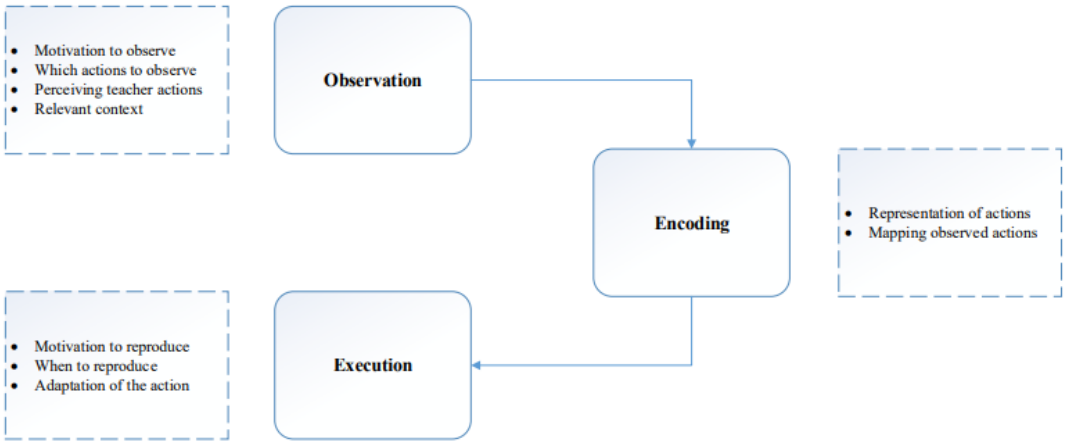


Figure 4.1

There are two basic main tasks in learning by imitation: a. Recognizing human behavior from visual input b. Finding methods to structure the motor control system for general movements and learning-by-imitation skills. Current approaches to representing a skill can be broadly divided into two trends: - Trajectory coding (low-level representation of skill in the form of a nonlinear mapping between sensory and motor information); - Symbolic coding (high-level representation of the skill that decomposes the skill into a sequence of action-perception units). To accomplish robot learning in response to a demonstration, three challenges must be addressed. 1. The correspondence problem is figuring out how to map connections and joints from human to robot. 2. Learning from demonstration is only feasible if a demonstrated movement can be generalized with the purpose of referring to various targets 3. Finally, robustness against perturbations is required: exactly reproducing an observed motion is not realistic in a dynamic environment, where obstacles may suddenly appear. A typical approach involves defining sufficient sub-skills and critical transition-assembly tasks: for modeling the assembly task, human demonstrations are translated into a sequence of movements referencing semantically relevant objects. Research in robotic assembly looks at specific developments: it requires a high degree of repeatability, flexibility, reliability to improve automation performance in assembly lines. Therefore, many specific research problems need to be solved to achieve automated robotic assembly in unstructured environments. The robot software should be able to convert assembly task sequences into individual movements, estimate the pose of assembly parts, and calculate the required forces and torques. To obtain an efficient assembly sequence for a task, an optimization algorithm is required to find the optimal

definition of the action to be performed. In some developments reported in the literature[13], researchers have found that the type of assembly has a significant influence on the choice of an optimal action sequence. Another such example considers an integrated assembly and motion-planning system to search for the assembly sequence with the help of a horizontal surface as a support. It proposes two new algorithms that learn precedence constraints and relative part size constraints. The first algorithm uses precedence constraints to generate assembly sequences in online mode and ensures the feasibility of assembly sequences by learning from human demonstration. The second algorithm learns how to assemble the various parts through exploratory executions, i.e., learning from exploration. Learning robots from demonstration requires the acquisition of example trajectories, which can be captured in various ways. Alternatively, a robot can be physically guided through the desired trajectory by its operator, and the learned trajectory is recorded proprioceptive for demonstration. To achieve a system equipped with proprioception, the following components must be considered: a robotic manipulator [5], a depth camera [6], image processing algorithms.

5. e.Do by Comau

The system used for the thesis project is e.DO Robot, product by Comau. This robot is a modular, multi-axis articulated (anthropomorphic with 6 degrees of freedom) Personal Care Robot. It has an integrated open-source intelligence; its aim is of making learning, creation, exploration and programming more fun and interactive.

e.DO is available in different versions and configurations:

- e.DO with 6 central axes
- e.DO with 6 side axes
- e.DO with 6 central axes, with Gripper
- e.DO with 6 side axes, with Gripper

During this project, the configuration of e.DO with Gripper is chosen.

The system used for the thesis project is e.DO Robot, product by Comau. This robot is a modular, multi-axis articulated (anthropomorphic with 6 degrees of freedom) Personal Care Robot. It has an integrated open-source intelligence; its aim is of making learning, creation, exploration and programming more fun and interactive.

e.DO is available in different versions and configurations:

- e.DO with 6 central axes
- e.DO with 6 side axes
- e.DO with 6 central axes, with Gripper
- e.DO with 6 side axes, with Gripper

During this project the configuration of e.DO with Gripper is chosen.

Specification	Value
Number of axis	6
Max payload	1 kg
Max reach	Axis 1: +/- 180° (38°/sec)
	Axis 2: +/- 113° (38°/sec)
	Axis 3: +/- 113° (38°/sec)
	Axis 4: +/- 180° (56°/sec)
	Axis 5: +/- 104° (56°/sec)
	Axis 6: +/- 180° (56°/sec)
Total weight	11,1 kg
Robot arm weight	5,4 kg

Structure material	Ixef 1002
Power source	Universal external power source with 12V power adapter
Connectivity	1 external USB port 1 RJ45 Ethernet 1 DSub-9 Serial Port
Motherboard	Raspberry Pi running Raspbian Jessie
ROS	Kinetic Kame
Control Logic	Proprietary open-source e.DO
Additional features	External emergency stop button

Table 4.1

Real measurements of each joint are reported in figure 4.2:

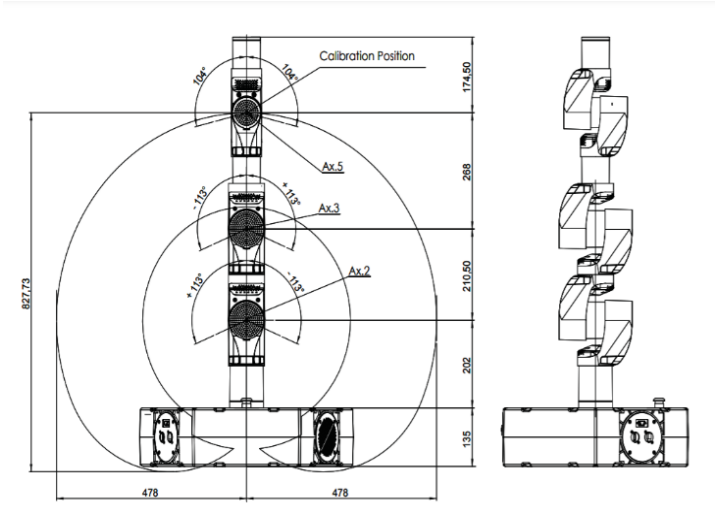


Figure 5.1

Main components

Open source

e.DO works thanks to ROS (Robot Operating System). This operating feature is the reason why people interesting in robotics can program on e.DO with different kind of programming language: C/C++, python, Java and the robot can interact with any hardware or software compatible with ROS.

Hardware and software

The hardware of e.DO robot is composed by a Raspberry Pi board and its operating system is Linux. The environment on which is developed makes possible that advanced users enter the embedded control system and improve it: this system provides intelligence and flexibility to realize complex movement, executions of sequence and automation in real world processes.

Main structure components

The Robot is characterized by a modular composition: “smallest construction units” are assembled and connected to obtained it. These units are described below:

1. The base of the Robot is a basic structure equipped with interface devices, inside with the Robot main power supply and controller boards are installed.
2. The Big joint unit consists of joint board (for motor control), motor, encoder, gearbox, brave, main shaft, support plate, gears and output flange. It constitutes the first 3 axes of the robot.
3. The Small joint unit consists of the same component of the Big one. It constitutes the remaining 3 axes of the robot.
4. There are brackets and adapters used for physical connection between the various joints.
5. The system presents an additional emergency push button, available to be positioned freely near the robot. The cap connector is available and can be used in case of absence of additional energy push-buttons and/or interlocking devices associated with guards.

The axes can rotate clockwise or anticlockwise. Directly on the robot structure are affixed some indicators “+” / “-”. Considering this labels, the direction of rotation is known and the identification of the axis is easily done.

5.1 Robot certification

The e.DO Robot is considered a “Machine” as defined in Article 2(a) of the Machinery Directive 2006/42/EC and anthropomorphic “Personal Care Robot (type 1.1)” with 6 degrees of freedom as set out in the standard EN ISO 13482:2014. The Robot complies with the requirements of the following Community Directives: – DIRECTIVE 2006/42/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 17 May 2006 on machinery, and amending Directive 95/16/EC.

The following are the harmonized standards followed for the design and construction of the Robot: – EN ISO 13482: 2014 Robot and robotic devices – Safety requirements for personal care Robots – EN ISO 12100: 2010 Safety of machinery - General principles for design - Risk assessment and risk reduction. – EN ISO 13849-1: 2016 Safety of machinery – Safety related parts of control systems – Part 1: General principles for design. – EN 60204-1: 2006 Safety of machinery - Electrical equipment of machines. Part 1: General requirements and subsequent amendments.

5.2 Main features:

1. Requirements: e.DO robot can be controlled through a special all that can be installed on tablets with Android operating system and minimum screen size of 7’’. The Wi-Fi connectivity requirement is 802.11 a/b/g/n/ac and the CPU must be Octa-core ((1.8 GHz Quad + 1.4 GHz Quad).
   After download and connection of e.DO app with e.DO robot, the configuration can be start.

2. Configuration: after the connection app-robot, it is possible to select e.DO configuration we will use (we use 'with Gripper' configuration).
   Then, at first start of e.DO, the calibration procedure is required. This is the process that must be followed (by using a specific app developed for):
   a. Using the buttons at the bottom, select the joint to be calibrated;
      NB: The e.DO calibration must be performed for all joints; it is suggested a progressive calibration from 1 to 6
      NNB: If the Robot is already in the calibration position (Robot in "vertical" position), it is possible to calibrate all the joints simultaneously by pressing "Calibrate all joints" button>
   b. Using "+/-" buttons, move the selected joint as long as the calibration notches on the Robot structure are aligned;
      NB: The axis "+/-" direction of rotation is indicated by special identification labels installed directly on the Robot structure. The axis movement speed can be adjusted through the slider in the center of the page
   c. repeat the calibration operations for the remaining axes of the Robot.

The calibration:

<u>is required</u>

   d. after turning off e.DO, setting the main switch to the 0-OFF position
   e. due to the lack of the electrical power supply to e.DO

<u>is not required</u>

   f. after pressing the emergency stop push-button
   g. after a collision

There is the possibility to calibrate the robot by using an SDK instead of the tablet application. This SDK is named 'pyedo' (https://github.com/Comau/pyedo) and contains some function to command the robot, starting from the initialization it is possible to calibrate and then control the movement of the robot. The function developed in this SDK are explained following:

- init_7Axes(): can initialize the robot with 7 axes (e.DO 6 axes plus the gripper)
- init_6Axes(): can initialize the robot with 6 axes
- disengage_std(): can disengage the robot with the standard movement
- disengage_safe(): can release only the brakes
- disengare_sin(): can disengage the robot with a sinusoidal movement
- calib_axes(): can calibrate all the robot axes. The calibration is possible only after initialization and disengage of the brakes
- move_joint(ovr, j1, j2, j3, j4, j5, j6, j7) : Can move the joints to a joint_position indicated through the angles in degrees (j1, , j2, j3, j4, j5, j6) and the opening of the gripper J7 in mm with a velocity percentage in (ovr) up to the maximum value «100»

- move_cartesian(ovr, x, y, z, a, e, r) : Can move the joints to a cartesian_position indicated through the pose in (x, y, z, a, e, r) with a velocity percentage in (ovr) up to the maximum value «100»
- move_circular(ovr, x1, y1, z1, a1, e1, r1, x2, y2, z2, a2, e2, r2) : Can move the joints to a cartesian_position indicated through the pose in (x1, y1, z1, a1, e1, r1) passing through another cartesian_position indicated through the pose in (x2, y2, z2, a2, e2, r2) with a velocity percentage in (ovr), up to the maximum value «100», creating a circular path.
- move_cancel() : Can cancel the buffer of the moves, helpful when some errors occur in generating path trajectory not allowed.
- move_cartesianX(ovr, x, y, z, a, e, r, j7) : Can move the joints to a cartesian_position indicated through the pose in (x, y, z, a, e, r) and the opening position in mm of the gripper in (j7) with a velocity percentage in (ovr) up to the maximum value "100"
- move_circularX(ovr, x1, y1, z1, a1, e1, r1,j71, x2, y2, z2, a2, e2, r2,j72) : Can move the joints to a cartesian_position indicated through the pose in (x1, y1, z1, a1, e1, r1) and the opening position in mm of the gripper in (j71) passing through another cartesian_position indicated through the pose in (x2, y2, z2, a2, e2, r2) and the opening position in mm of the gripper in (j72) with a velocity percentage in (ovr), up to the maximum value "100", creating a circular path.
- jog_joint(ovr, j1, j2, j3, j4, j5, j6, j7) : Can move the joints, once per time for a delta, in a direction indicated with "1"(positive) or "-1"(negative) in the (j1, j2, j3, j4, j5, j6, j7) with a velocity percentage in (ovr) up to the maximum value "100"
- jog_cartesian(ovr, x, y, z, a, e, r) : Can move the joints in a cartesian_position, once coordinate per time for a delta, indicated with "1"(positive) or "-1"(negative) in the cartesian_position through the pose in (x1, y, z, a, e, r) with a velocity percentage in (ovr) up to the maximum value "100"
- listen_JointState() : Allows to start the subscribing on the topic related to this information: [Position Velocity Current]
- listen_CartesianPosition() : Allows to start the subscribing on the topic related to the cartesian position (x,y,z,a,e,r)
- unlisten_JointState() : Allows to stop the subscribing on the topic related to these information: [Position Velocity Current]
- unlisten_CartesianPosition() : Allows to stop the subscribing on the topic related to the cartesian position (x,y,z,a,e,r)
- get_JointState() : Allows to get the Joint State dictionary containing the variables listened with the listen method: [Position Velocity Current cartesianPosition]

The procedure of connection is the same as above, but the initialization, calibration and moving are commanded by these functions. In the Thesis project this approach has been adopted. First of all the robot is powered on, then connection between pc and robot is performed using Wi-Fi. Once the connection is stable, a few steps are performed as reported below:

- Open a new terminal and digit:
  a. ssh edo@192.168.12.1 (and the password)

b. export LC_ALL=C; unset LANGUAGE
c. .\ministarter_sim

When the state of the robot is “READY” open an offline python script and use it to send the commands to the robot. The example of this code is reported below.

```
from pyedo import edo
myedo= edo('192.168.12.1')
import time
myedo.move_joint()
myedo.init_7Axes()
time.sleep(7)
myedo.disengage_std()
time.sleep(7)
myedo.calib_axes()
```

A break point is added at the end in order to have the possibility to send single command a time by using ‘evaluate’ tool.

After calibration command, we must be sure that the robot is in the “vertical position”: there are some signs on each joint and the lower one must fit with the upper one in order to be sure that the position is the suggested one.

The fitting between sign is reached by using move_joint(), giving in input the angular position necessary to reach the best fitting of the signs.

Then, recalibration is done through ‘calib_axes()’ after the robot is in "vertical position" in these positions.

Now the robot is ready to receive in input the angle position for each joint: this kind of control is based on the direct kinematics.

5.3 Inverse kinematics problem

The motion control based on inverse kinematics solver can be handled by using the ‘move_cartesian’ function of the SDK. Unfortunately, this function is not working how we expected, the reason of this failure can be that it uses an inverse kinematic solver to operate, and this solver is not present inside the robot as default. In order to solve this problem, an analysis on inverse kinematic solver has been done. Considering that the project is based on a python programming, we have searched for python inverse kinematic solvers. The idea is to

solve the inverse kinematic, giving as input the Cartesian coordinate of the target and extracting from the output the parameters for move_joint() function (that is based on direct kinematics).

The first solver used to obtain this process is named 'tinyIK'. It is based on a very simple logic and works on a chain created by the user. As first step, a model of the robot must be defined:

```
import tinyik
arm = tinyik.Actuator(['z', [1., 0., 0.], 'z', [1., 0., 0.]])
print(arm.angles)
print(arm.ee)
import numpy as np
arm.angles = [np.pi / 6, np.pi / 3]
print(arm.ee)
arm.ee = [2 / np.sqrt(2), 2 / np.sqrt(2), 0.]
print(np.round(np.rad2deg(arm.angles)))
```

Through the function chain, we can define a very simple model. It is enough to declare the rotation axis of each joint and relative length. Then, the library offers some function, we use directly the inverse_kinematic function to solve our model trajectory planning. The output of that function is the sequence of joint position angles that the robot has to follow. By creating a code in which pyedo and tinyIK works together, it is possible to obtain the solution to the inverse kinematic problem.

The model created on tinyIK is very simple and ignoring some of important characteristics of the robot (inertia, joint limits in motion, orientation).

For this reason, another python inverse kinematic solver is tested.

In order to bypass the RViz and MoveIt computations and implement the same logic in a smarter way, an alternative Inverse Kinematic solver is used: IKPy.

With IKPy, it is possible to:

- Compute the Inverse Kinematics of every existing robot.
- Compute the Inverse Kinematics in position, orientation, or both
- Define your kinematic chain using arbitrary representations: DH (Denavit–Hartenberg), URDF, custom...
- Automatically import a kinematic chain from a URDF file.
- Use pre-configured robots, such as baxter or the poppy-torso
- IKPy is precise (up to 7 digits): the only limitation being your underlying model's precision, and fast: from 7 ms to 50 ms (depending on your precision) for a complete IK computation.
- Define your own Inverse Kinematics methods.IKPy is an inverse kinematic solver that solve the kinematic of robots on a model described by the user or imported through the URDF file. The URDF file contains more details about each important feature of the robot and for this reason the analysis results are more precise than in previous situation.

The URDF file of e.DO robot is reported below: it is the mix between the model created by Comau (https://github.com/Comau/eDO_description) and the model of a project developed by some Reply employees (https://gitlab.com/aloha-reply).

```
<robot
  name="edo_sim">
  <link
    name="base_link">
    <inertial>
      <origin
        xyz="0.0617583602130883 0.437262464550775 -
0.00395442197852672"
        rpy="0 0 0" />
      <mass
        value="0.0785942338762368" />
      <inertia
        ixx="0.0123841200738068"
        ixy="-0.000187984913202787"
        ixz="-1.32683892634308E-06"
        iyy="7.0169034503364E-05"
        iyz="-9.17416945099319E-05"
        izz="0.0123862261905614" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/base_link.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/base_link.STL" />
      </geometry>
    </collision>
  </link>
  <link
    name="link_1">
    <inertial>
      <origin
        xyz="-0.00457048841401064 0.303831811417004 -
0.00202866410579916"
        rpy="0 0 0" />
      <mass
        value="0.0785942338762368" />
      <inertia
        ixx="0.0123841200738068"
        ixy="0.000187984913202727"
```

```
          ixz="-1.32683892634271E-06"
          iyy="7.01690345033622E-05"
          iyz="9.17416945099368E-05"
          izz="0.0123862261905615" />
      </inertial>
      <visual>
        <origin
          xyz="0 0 0"
          rpy="0 0 0" />
        <geometry>
          <mesh
            filename="package://edo_sim/meshes/link_1.STL" />
        </geometry>
        <material
          name="">
          <color
            rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
        </material>
      </visual>
      <collision>
        <origin
          xyz="0 0 0"
          rpy="0 0 0" />
        <geometry>
          <mesh
            filename="package://edo_sim/meshes/link_1.STL" />
        </geometry>
      </collision>
    </link>
    <joint
      name="joint_1"
      type="continuous">
      <origin
        xyz="0.057188 0.0059831 0.13343"
        rpy="1.5708 6.9389E-16 -3.1416" />
      <parent
        link="base_link" />
      <child
        link="link_1" />
      <axis
        xyz="0 1 0" />
    </joint>
    <link
      name="link_2">
      <inertial>
        <origin
          xyz="-0.0168406485709407 0.071318237296396 -
0.0876822373080704"
          rpy="0 0 0" />
        <mass
          value="0.0785942338762368" />
        <inertia
          ixx="0.0122070242873091"
          ixy="0.000930596667670934"
          ixz="-0.0011486325666074"
          iyy="0.007576065624266"
          iyz="0.0059380539105297"
          izz="0.00505742538729646" />
      </inertial>
      <visual>
```

```
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_2.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_2.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="joint_2"
    type="continuous">
    <origin
      xyz="0 0.18967 0"
      rpy="0.94237 -0.4634 -0.11653" />
    <parent
      link="link_1" />
    <child
      link="link_2" />
    <axis
      xyz="-0.88847 0.2908 0.35504" />
  </joint>
  <link
    name="link_3">
    <inertial>
      <origin
        xyz="0.00457048841401063 0.0962524890074447
0.00395442197852662"
        rpy="0 0 0" />
      <mass
        value="0.0785942338762368" />
      <inertia
        ixx="0.0123841200738068"
        ixy="0.000187984913202718"
        ixz="1.32683892634266E-06"
        iyy="7.01690345033619E-05"
        iyz="-9.17416945099381E-05"
        izz="0.0123862261905615" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_3.STL" />
```

```
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_3.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="joint_3"
    type="continuous">
    <origin
      xyz="-0.024558 0.12737 -0.16578"
      rpy="0.97336 -0.36296 2.8253" />
    <parent
      link="link_2" />
    <child
      link="link_3" />
    <axis
      xyz="1 0 0" />
  </joint>
  <link
    name="link_4">
    <inertial>
      <origin
        xyz="-0.00422951158597114 -0.00395442197852627
0.255057059248813"
        rpy="0 0 0" />
      <mass
        value="0.0785942338762368" />
      <inertia
        ixx="0.0123841200738068"
        ixy="-1.32683892634285E-06"
        ixz="0.000187984913202718"
        iyy="0.0123862261905615"
        iyz="9.17416945099506E-05"
        izz="7.01690345033621E-05" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_4.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
```

```
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_4.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="joint_4"
    type="continuous">
    <origin
      xyz="0.0088 -0.1588 0"
      rpy="-1.5708 0 0" />
    <parent
      link="link_3" />
    <child
      link="link_4" />
    <axis
      xyz="0 0 -1" />
  </joint>
  <link
    name="link_5">
    <inertial>
      <origin
        xyz="0.00422951158596777 -0.00395442197852616 -
0.360352489007445"
        rpy="0 0 0" />
      <mass
        value="0.0785942338762368" />
      <inertia
        ixx="0.0123841200738068"
        ixy="1.32683892634191E-06"
        ixz="0.000187984913202582"
        iyy="0.0123862261905615"
        iyz="-9.17416945099552E-05"
        izz="7.01690345033581E-05" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_5.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333
1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
```

```
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_5.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="joint_5"
    type="continuous">
    <origin
      xyz="0 0 -0.1053"
      rpy="3.1416 1.1102E-14 3.1416" />
    <parent
      link="link_4" />
    <child
      link="link_5" />
    <axis
      xyz="-1 0 0" />
  </joint>
  <link
    name="link_6">
    <inertial>
      <origin
        xyz="1.10581233290358E-05 -0.009323393856032096.35624315718574E-06"
        rpy="0 0 0" />
      <mass
        value="0.0279702497322662" />
      <inertia
        ixx="7.63464744598253E-06"
        ixy="1.89204959067221E-10"
        ixz="5.32970316039599E-09"
        iyy="1.45744912344428E-05"
        iyz="1.08958003890421E-10"
        izz="7.62871791498103E-06" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_6.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.9333333333333331" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://edo_sim/meshes/link_6.STL" />
      </geometry>
    </collision>
  </link>
```

```
    <joint
      name="joint_6"
      type="continuous">
      <origin
        xyz="-0.0039 0 0.1636"
        rpy="-1.5708 1.249E-14 0" />
      <parent
        link="link_5" />
      <child
        link="link_6" />
      <axis
        xyz="0 -1 0" />
    </joint>
  </robot>
```

Through this model is described each joint and each link (end effector is not considered[1]), including the base link (that is fixed). Each link is described in terms of origin (of relative reference frame), orientation (of relative RF), inertia, mass, speed. Each joint is described in terms of origin (of relative RF), orientation, rotation axis, joint limits (the range of the angles that can be performed), speed limits. In other words, the links describe static features and the joints include features that makes possible the motion control.

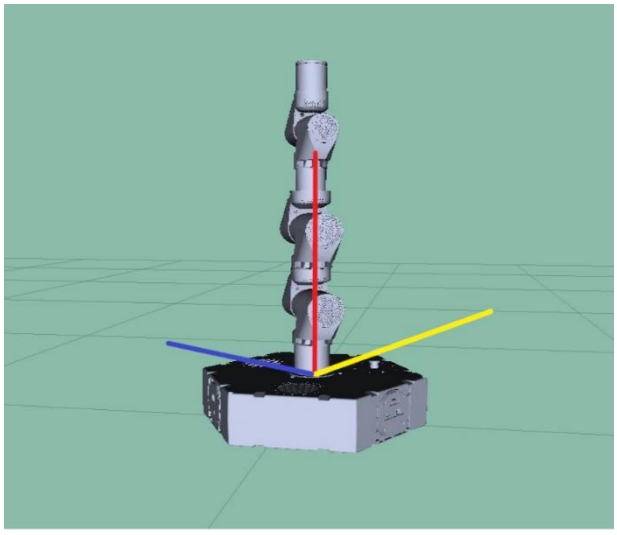


Figure 5.2: URDF model on RViz (visualization software)

After creation of the model in URDF format, it is possible to import it in the 'chain' variable, through the use of *Chain* function.

'link' is specified as 'base_element_type' and the model is ready to be used for the simulation.

Once the model is imported is possible to visualize a minimal structure representing the robot, using the *matplotlib* library functions. The model can be verified by showing on a plot the structure of the defined chain. Each joint shows its rotation axis exiting so is very intuitive to understand if there are errors in the model. The results of the model created is reported below:

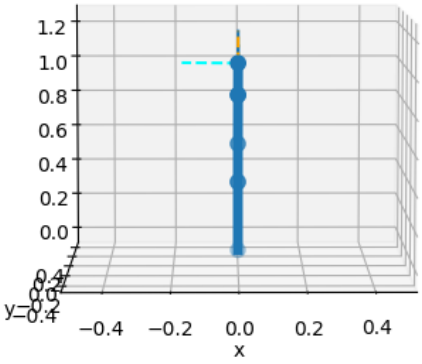


Figure 4.4: xz plane view

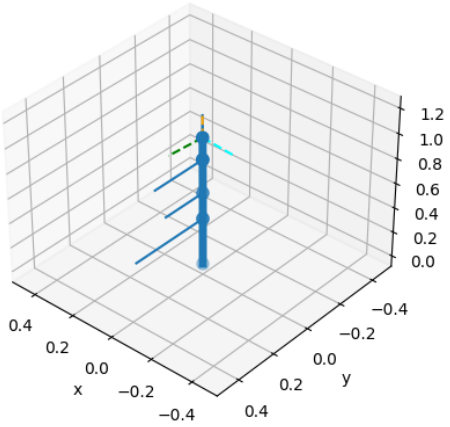


Figure 4.5: xy plane view

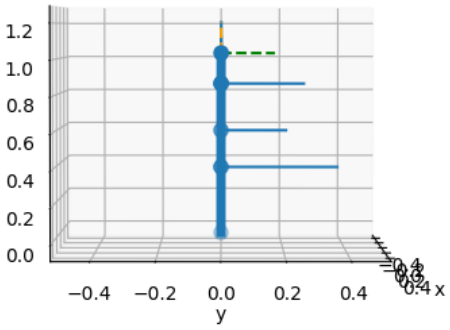


Figure 4.6: yz plane view

Now it is possible to demonstrate the logic of the inverse kinematic solver on which is based IKPy solver. The syntax of the function solver of inverse kinematics is:

Input (there are other inputs than these below maintained at default value):

- *TARGET_POSITION*: is the position in coordinates of the point that the robot has to reach. The values have as reference the length of joints described by the model.
- *TARGET ORIENTATION*: is a unit vector, it defines the axis along the orientation axis will be aligned. It is defined by the cosines of the angle between the vector and each reference system axes.
- *ORIENTATION_MODE*: is the axis of the last joint reference system that will be aligned with the unit vector defined in target_orientation.

Output:

- angles for each link (7), of which the first one is zero because the joint is modelled as fixed. The other angles, if different from zero, have value limited in the range defined in the URDF file.

The end effector is not considered in the model, due to the fact that it is not a real joint, instead it is composed by more than one mobile part, and it is not possible to define it as revolute or planar joint, just fixed. Considering that the logic of the orientation control is based on alignment of the last defined joint rotation axis with the unit vector defined in target_orientation, the end effector makes the model ambiguous in terms of orientation control. In fact, the orientation of the end effector depends on the Joint 6 orientation (if we consider the EE as a fixed joint). Since this fact we want the J6 will align its reference frame how we prefer and to obtain this Joint 6 must be seen as the last joint.

When all the parameters are defined, it is possible verifying the angles in output on a plot that shows a very minimal model. Some tests are performed to be evident the logic on which is based the solver. First, the 'zero' position is showed, giving to the direct kinematic function all angles set to zero.

An example of the algorithm is reported following:

```
import ikpy
import ikpy.inverse_kinematics
import numpy as np
#from pyedo import edo
#import time
import ikpy.utils.plot as plot_utils

#kinematic Chain
edosim =
ikpy.chain.Chain.from_urdf_file("/home/anna/catkin_ws/src/eDO_comau/robots/
edited_edo_sim.urdf")

#getting the position of your chain
positionchain=edosim.forward_kinematics([0] * 7)
print("posizione ad angoli nulli: ", positionchain[:3,3])

newpos=[0.2,6.58959642e-04,1]
print('new pos ', newpos)
newangle1=edosim.inverse_kinematics(newpos)
newangle=np.round(np.rad2deg((edosim.inverse_kinematics(newpos))))
print("angoli per raggiungere newpos: ",newangle)
```

```
positionangle=edosim.inverse_kinematics([7.04527514e-04, 6.58959642e-04,
1.02999907e+00])
print("angoli per raggiungere la posizione zero:
",np.round(np.rad2deg(positionangle)))


import matplotlib.pyplot
from mpl_toolkits.mplot3d import Axes3D
ax=matplotlib.pyplot.figure().add_subplot(111, projection='3d')

edosim.plot(positionangle, ax)
edosim.plot(newangle1, ax)
matplotlib.pyplot.show()


#myedo = edo("192.168.12.1")
#myedo.init_7Axes()
#time.sleep(1)
#myedo.disengage_std()
#time.sleep(5)
#myedo.calib_axes()
#time.sleep(1)

#myedo.move_joint(j1=0,j2=0,j3=0,j4=0,j5=34,j6=0)
```

6. Stereo-vision and depth acquisition

Human vision is one example of how the stereo-vision works. The perception of depth is due to our brain's ability in analyzing the difference between the two-dimensional images which are projected of the retinas of the eyes. To extract 3-D information from a given scene, multiple camera views of the same scene are required. These images can be captured using either a single movable camera of an array of cameras. The depth of the real-world scenery can thus be estimated by comparing the difference between the left and tight digital images captured by each camera.

The two key aspect of computer stereo-vision are speed and accuracy. A lot of research has been carried out to improve both the precision disparity maps and the execution speed of the algorithm. Speed and precision are two desirable but conflicting properties, it is very challenging to achieve both simultaneously. The algorithm execution can be improved with future advances in hardware computational power.

6.1 Geometry and computer vision in stereo-vision: perspective Camera model

The perspective (or pinhole) camera model is the most commonly used geometric model to describe the relationship between a 3D point $p = [X, Y, Z]^T$ in the Camera Coordinate System (CCS) and its projection $p = [x, y, z]^T$ on the image plane π. An example of the perspective camera model is shown in Figure 20 where *o* is the focus of the camera and the distance between π and *o* is the focal length *f*.

$\underline{p} = [X/Z, Y/Z, 1]$ is an image point expressed in normalized coordinates. The ray originating from *o* and going perpendicularly through π is known as the optical axis. $\underline{o} = [o_u, o_v]'$ and the intersection between π and the optical axis is the principal point in pixels. Since the distance z between *o* and the image plane π is always equal to the focal length f, the relationship between *p* and $\underline{p}$ is shown as:

eq.1
$$\bar{p} = \frac{f}{Z^l} p^l$$

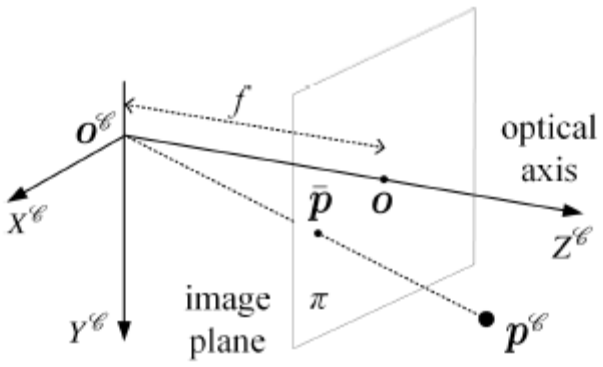


Figure 6.1

Intrinsic parameters

Since the lens distortion does not exist in a perspective camera model, the transformation from a projection point $\underline{p}$ on the image plane to a pixel $p = [u, v]^T$ in the Image Coordinate System (ICS) is performed as follows:

eq.2
$$u = O_u + s_x x$$
$$v = O_v + s_y y$$

where s_x and s_y are the effective size measured in pixels per millimetre in the horizontal and vertical directions, respectively. To simplify the expression of the intrinsic matrix K, two notations $f_x = fs_x$ and $f_y = fs_y$ are introduced. Combining eq.1 and eq.2, a 3-D point p in the CCS can be transformed to a pixel p in the ICS using Eq.3, where $\tilde{\mathbf{p}} = [\boldsymbol{p}^T; 1]^T = [u, v, 1]^T$ denotes the homogeneous coordinate of $\boldsymbol{p} = [u, v]^T$. It is to be noted that an arbitrary 3-D point lying on the ray which goes from $\boldsymbol{o}^c$ and through $\boldsymbol{p}^c$ is always projected at $\boldsymbol{p}$ in the ICS. o_u, o_v, f, s_x and s_y are five intrinsic parameters.

eq.3
$$z_{\tilde{p}} = Kp^l = \begin{bmatrix} fx & 0 & O_u \\ 0 & fy & O_u \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X^l \\ Y^l \\ Z^l \end{bmatrix}$$

Given the matrix K (intrinsic matrix) an image point can be expressed in normalized coordinates as follows:

eq.4
$$\hat{p} = K^{-1}\tilde{p} = \frac{\bar{p}}{f} = \frac{p^l}{z^l}$$

Lens distortion To get a better image result, a lens is usually installed in front of the camera, but it introduces distortion into the images. Optical aberration caused by the lens distorts physically straight lines and makes them appear as curves in the images. Lens distortions can be grouped into two main categories: radial and tangential. The presence of radial distortion is due to the fact that the geometric shape of the lens affects the transmission of straight lines, while tangential distortion occurs because the lens installed in front of the camera is not perfectly parallel to the image plane. In practical experiments, image geometry is affected to a much greater extent by radial distortion than by tangential distortion, and so the latter is always neglected when correcting a distorted image. For calibration of a multi-camera system, correction of lens distortion is usually performed before estimating the intrinsic and extrinsic parameters. Radial distortion can be classified primarily in two ways (example shown in figure 22). It can be observed that radial distortions are symmetrical around the center of the image, and the lines are no longer straight in distorted images. In barrel distortion, the magnification of the image decreases with distance from the center of the image (the lines curve outward). In contrast to barrel distortion, pincushion distortion pinches the image (the lines curve inward). Fortunately, these two types of radial distortion can be corrected using eq.5, where $p = [u, v]^T$ is

a pixel in the distorted image and $p_{corrected} = [u_{corrected}; v_{corrected}]^T$ is the displacement of p in the corrected image.

- $r = \|p - o\|^2$ is the distance from p to the center of the image.
- k1, k2, and k3 are three intrinsic parameters used to correct radial distortion, and can be estimated using different checkerboard images.

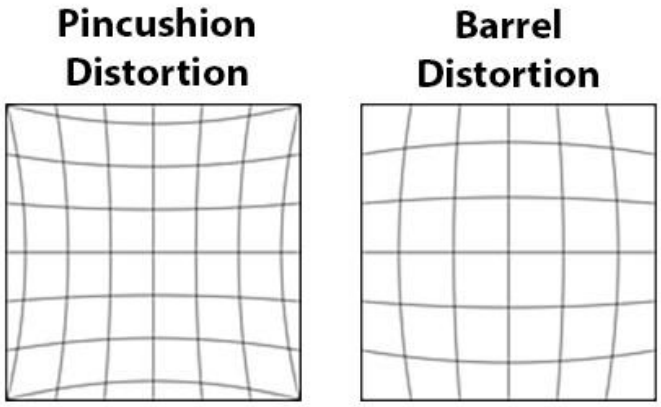


Figure 6.2

eq.5

$$u_{corrected} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)(u - o_u) + o_u$$
$$v_{corrected} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)(v - o_v) + o_v$$

Tangential distortion

Similar to radial distortion, the tangential distortion can be corrected, using intrinsic parameters which can be estimated using several images containing a planar checkerboard pattern.
In practical experiments, radial and tangential distortions are usually corrected simultaneously. the corresponding correction result of Figure 6.3 (a) is shown in Figure 6.3 (b), where the bent checkerboard grids are now displayed linearly.

Figure 6.3

The generic geometry of a binocular vision system is known as epipolar geometry. An example of epipolar geometry is shown in Figure 6.4:

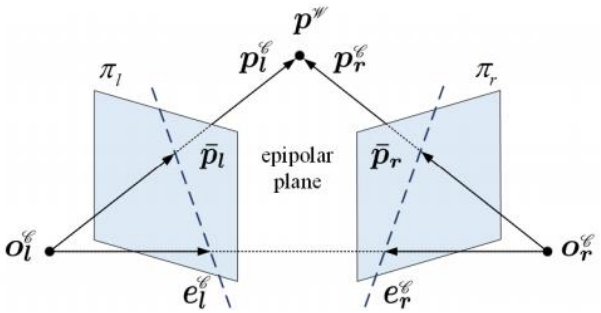


Figure 6.4

- O_l^c and O_r^c denote the focuses of the left and right cameras, respectively.
- $p^w = [X^N, Y^w, Z^w]$ is a 3-D point in the World Coordinate System (WCS) and its representations in the Left Camera Coordinate System (LCCS) and Right Camera Coordinate System (RCCS) are $p^c_L = [X^c_L ; Y^c_L; Z^c_L]^T$ and $p^c_R = [X^c_R ; Y^c_R ; Z^c_R]^T$, respectively.
- Π_L and Π_R are two image planes. P^W is projected on Π_L at $p_L = [x_L; y_L; f_L]^T$ and on Π_R at $p_R = [x_R; y_R; f_R]^T$, where f_L and f_R are the focuses of the left and right cameras, respectively. e^c_L and e^c_R denote the left and right epipoles, respectively.

The plane identified by o^c_L, o^c_L and p^W is known as an epipolar plane. The latter intersects each plane in a line which is generally named as an epipolar line. The transformation from a 3-D point in each CCS to its projection on the corresponding camera plane can be performed using eq.6:

eq.6
$$\overline{p_l} = \frac{f_l}{z_l^C} p_l^C$$
$$\overline{p_r} = \frac{f_r}{z_r^C} p_r^C$$

$p_l^C \; and \; p_r^C$ can be normalized as:

eq.7
$$\hat{p}_l = \frac{p_l^C}{z_l} = K_l^{-1} \tilde{p}_l$$
$$\hat{p}_r = \frac{p_r^C}{z_r} = K_r^{-1} \tilde{p}_r$$

where K_l and K_r denote the intrinsic matrices of the left and right cameras, respectively. $\tilde{p}_l = [\tilde{p}_l^T; 1]^T$ and $\tilde{p} = [\tilde{p}_r^T; 1]^T$ represent the homogeneous coordinates of the 2-D points $p_l = [u_l; v_l]^T$ and $pr = [u_r; v_r]^T$, respectively. Before moving on to analyze the extrinsic parameters of an

epipolar geometry, it is introduced a matrix R Є R^{3x3} and a translation vector t Є R^{3x1} to describe the transformation from p_l^C to p_r^C (eq.8).

eq.7
$$p_r^C = Rp_l^C + t$$

Estrinsic parameters

E depicts the relationship between each pair of normalized image points p_l and p_r lying on the same epipolar plane. It is important to note here that E has five degrees of freedom (both R and t have three degrees of freedom, but the overall scale ambiguity causes the degrees of freedom to be reduced by one). Hence, in theory, E can be estimated with at least five pairs of p^c_l and p^c_r.

eq.8
$$\hat{p}_r^T E \hat{p}_l = 0$$

However, due to the non-linearity of E, its estimation using five pairs of correspondences is always intractable. Therefore, E is commonly estimated with at least eight pairs of p^c_l and p^c_r. The essential matrix creates a link between each pair of corresponding 3-D. When the intrinsic matrix of each camera is known, the relationship between each pair of corresponding 2-D points $p_l = [u_l; v_l]^T$ and $p_r = [u_r; v_r]^T$ can also be established. This process relates to a so-called fundamental matrix. It can be thought of as a generalization of the essential matrix, where the assumption of calibrated cameras is removed. The fundamental matrix **F** Є $\mathbf{R}^{3x3}$ is defined as:

eq.9.1
$$F = K_r^{-T} E K_l^{-1}$$

eq.9.2
$$\tilde{p}_r^T K^{-T} E K_l^{-1} p_l = \tilde{p}_r^T F \tilde{p} l = 0$$

F (Eq. 9.1) has seven degrees of freedom: five come from E (Eq. 8) and the other two from K_r and K_l . The most commonly used algorithm for estimating E and F is the so-called "eight-point algorithm". This algorithm is based on the scale invariance of E and F, i.e., $\lambda_E p_r^{cT} E p_l^c = 0$ and $\lambda_F p_r^{cT} F p_l^c = 0$, where λ_E and λ_F are different from zero. By setting an element in E and F to 1, there are eight unknown elements that need to be estimated, and this can be done using at least eight matching pairs. If the intrinsic matrices K_l and K_r of the two cameras are known, the eight-point algorithm only needs to be run once to estimate E or F, because the other can be easily processed using the relationship between them.

6.2 Stereo rectification

When using a pair of pinhole cameras to acquire images from multiple views, the main task of 3D reconstruction is to determine each pair of corresponding points between the left and the right images. For a not calibrated stereo vision system, finding the correspondence pairs usually involves a 2-D search, with is a computationally intensive task. Therefore, an image

transformation process known as Stereo Rectification is always performed beforehand to reduce the dimension of the correspondence search. Each pair of conjugate epipolar lines becomes collinear and parallel to the horizontal image axis, as shown in figure 6.5:

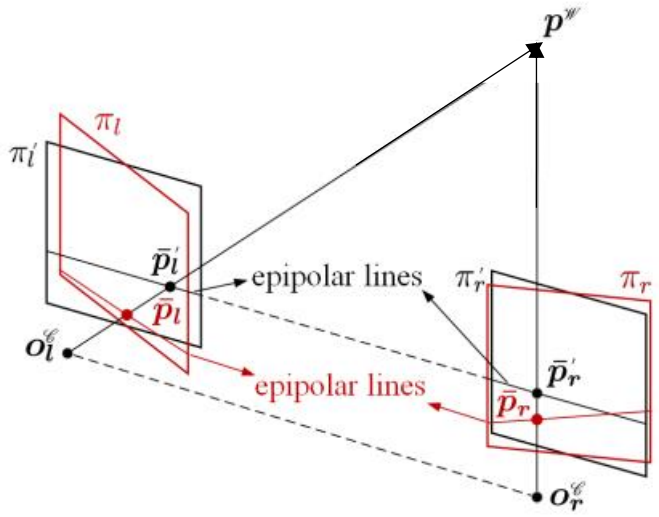


Figure 6.5

Where π_L' and π_R' are rectified image planes. After the rectification process, the left and right images appear as if they were taken using a pair of parallel cameras. Hence, searching for the correspondence pairs is simplified to a one dimensional (1-D) process.[14]

6.3 Basic Stereo Vision Model

A well-calibrated binocular system can be represented by a basic stereo vision model, as shown in figure:

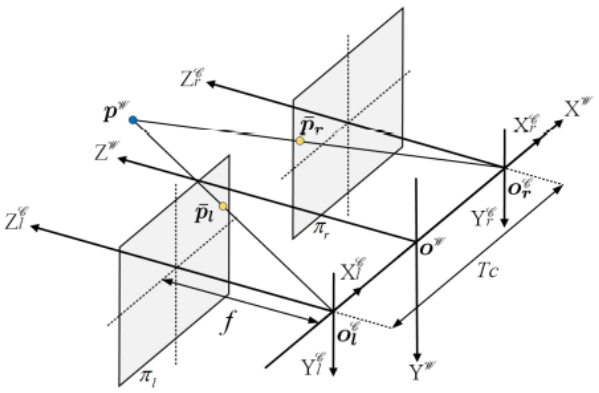


Figure 6.6

The latter can be regarded as a specialization of the epipolar geometry, where the left and the right cameras are perfectly parallel to each other and the axis X_l^c and the axis X_r^c are collinear.[15]

6.4 Approximation of theoretical entities

In general the relationship between a disparity (d) and depth (z) can be parametrized as seen in equation:

$$z = \frac{f * B}{d}$$

Where f is the focal length of the imaging sensor, B is the desired depth units.

Additionally, the derivative $\frac{\partial z}{\partial d}$, substituted in the equation generates an error:

$$|e1\,| = \frac{z^2}{f * B} * |e2\,|$$

Where $e1$ is the error on depth and $e2$ is the error on the disparity. It is usually constant for stereo system. It stems from imaging properties and the quality of the matching algorithm, so $|e2\,|$ is treated as a constant.

Classical stereoscopic depth systems struggle with resolving depth on texture-less surfaces. A plethora of techniques have been developed to solve this problem, from global optimization method to semi-global propagation techniques, to plane sweeping methods.

These techniques all depend on some prior assumptions about data in order to generate correct depth candidates. In the Intel RGBD depth cameras there is instead an active texture projector available on the module. This technique was used by classical stereo systems.

Such systems do not require a priori knowledge of the pattern's structure, as they are used simply to generate texture which makes image matching unambiguous.

To create a favorable pattern for optical configuration and matching algorithm, one can perform optimization over a synthetic pipeline that models a projector design. By modeling both the optical and the physical system's constraints and realistic imaging noise, one can obtain better texture projectors.

6.5 Depth camera: Intel RealSense D435i

A depth camera is a kind of sensor that can directly collect distance information between an object and the camera. The RealSense D435i is a low-cost depth camera that is currently in widespread use. Data acquisition produces two images simultaneously: RGB image and depth images. The quality of RGB image is good, whereas the depth image typically has many holes.[16]

Figure 6.7: Depth camera Intel RealSense D435i

The Intel RealSense D345i (shown in figure) places an IMU into our cutting-edge stereo depth-camera. With an Intel module and vision processor in a small form factor, this camera is a powerful complete package which can be paired with customizable software for a depth camera that is capable of understanding its own movement. The D435i is an ideal camera for motion application, the combination of a wide field of view and global shutter sensor on the camera make it the preferred solution for application such as robotic navigation and object recognition. The wider field of view allows a single camera to cover more area, resulting in less "blind spots". The global shutter sensors provide great low light sensitivity, allowing robots to navigate spaces with the light off.[17]

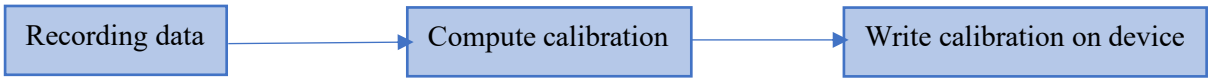
Setup:

The hardware required includes the D435i device to be calibrated, a USB cable, and a computer running Windows or Ubuntu. Inter RealSense SDK works can be used through python language, a python wrapper is available and open-source. The calibration script is included in the SDK available on GitHub (table below).

Resource	**URL**
SDK homepage	https://www.intelrealsense.com/developers#downloads
LibRealSense Githhub	https://github.com/IntelRealSense/librealsense
SDK documentation	https://github.com/IntelRealSense/librealsense/tree/master/doc
Python wrapper	https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python

Calibration Device with the Python Calibration Script:

The general process to calibrate a device with the Python Calibration Script starting the script to capture IMU data in 6 different positions, then computing the parameters and writing the result to the camera.



- Main features:

Use environment: Indoor/Outdoor

Image sensor technology: Global Shutter

Ideal range: 0.3 m to 3 m

- Depth features:

Depth technology: Active IR Stereo

Minimum depth distance (Min-Z) at max resolution: about 28 cm

Depth Accuracy: <2% at 2 m1

Depth Field of View (FOV): 87° × 58°

Depth output resolution: Up to 1280 × 720

Depth frame rate: Up to 90 fps

RGB frame resolution: 1920 × 1080

RGB frame rate: 30 fps

RGB sensor technology: Rolling Shutter

RGB sensor FOV (H × V): 69° × 42°

RGB sensor resolution: 2 MP

- Main Components:

Camera module: Intel RealSense Module D430 + RGB Camera

Vision processor board: Intel RealSense Vision Processor D4 Physical

Form factor: Camera Peripheral

Length × Depth × Height: 90 mm × 25 mm × 25 mm

Connectors: USB-C* 3.1 Gen 1*

Mounting mechanism:
- One 1/4-20 UNC thread mounting point.
- Two M3 thread mounting points.

In order to calibrate the system some distances are taken into account:

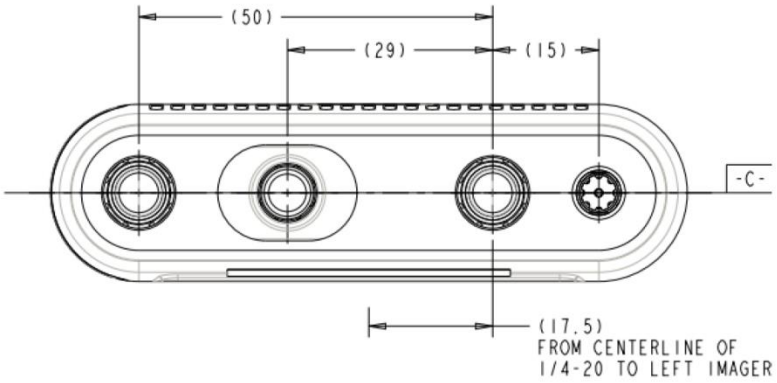


Figure 6.8: measures of d-camera

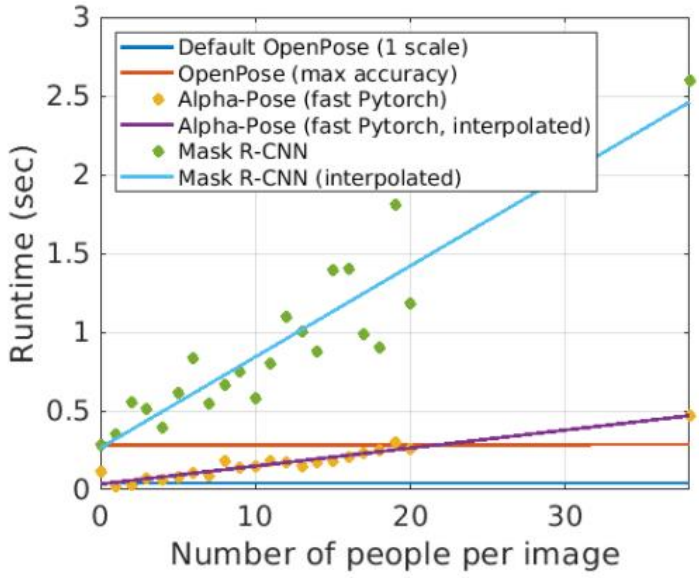
During tests, the RealSense camera is employed to acquire human gestures. Each image (RGB and depth) is used for the analysis. As first, the RGB image is used to compute two-dimensions movement; depth image information has been integrated successfully in order to transform the movement from 2D to 3D.

Deep learning techniques are used to process the images. A comparison with classified human poses is performed, using two neural networks pretrained. In order to optimize the performance, a library of git with a pre-existing class of images and trained neural network is chosen. In the next chapter, the library is described deeply.

7. Image processing: Openpose

OpenPose has represented the first real-time multi-person system to jointly detect human body, hand, facial and foot key points (in total 135 key points) on a single image.

Runtime analysis: in the graph below is shown the comparison between 3 available pose estimation libraries (assuming the same hardware conditions): Openpose, Alpha-Pose (fast Pytorch) and Mask R-CNN. The openpose runtime is constant, while the runtime of Alpha-Pose and Mask R-CNN grow linearly with the number of people. It is clear that openpose has better performance in case of multi detection.



Graph 7.1: comparison between CNN-based library

Main capabilities:

1. **2D real-time** multi-person key point detection:

   - 15,18 or 25 key points, including 6 foot key points. Runtime is invariant to number of detected people.

   - 2x21 key point hand estimation: runtime depends on number of detected people.

   - 70-keypoint face estimation: runtime depends on number of detected people.

2. **3D real-time** single person key point detection:
   - 3D triangulation from multiple single views
   - Synchronization of Flir cameras handled
   - Compatible with Flir/Point Grey cameras.

   Calibration (3D) toolbox: estimation of distortion, intrinsic and extrinsic camera parameters.

How it works:

Input can be an image or video, capturing real time by webcam, Flir/Point Grey, IP camera, and support to add your own custom input source (i.e. depth camera figure 28)

Output is input frame + keypoints get out from the elaboration; keypoints can be saved in file (JSON, XML, YML).

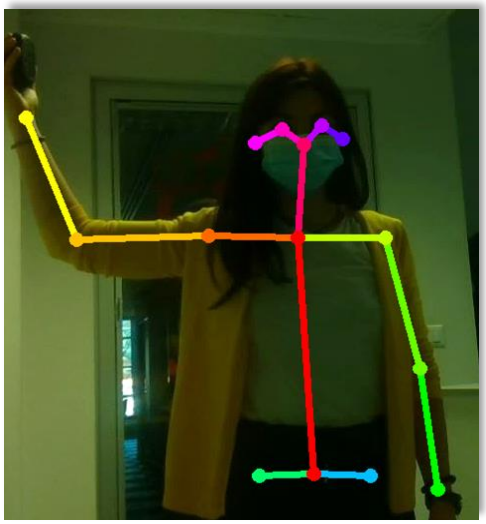


Figure 7.1: output OpenPose elaboration

Operating systems: Ubuntu (20,1816,14), Windows (10,8), MaxOSX, Nvidia TX2.

Hardware compatibility: CUDA (Nvidia GPU), OpenCL (AMD GPU), and non-GPU (CPU-only) versions.

In the specific case of this project, GPU is not compatible with the requested one. Until now, only-CPU version is used and the implementation is done in non-real time. Output skeleton is produced in some seconds (time to elaborate the image can range from 12 to 50 sec).

In order to accomplish to this requirement during the test I tried to launch Amazon services EC2 instances, creating it with the required hardware.

7.1 Amazon AWS

Amazon Web Services offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing. From data warehousing to deployment tools, directories to content delivery, over 175

AWS services are available. New services can be provisioned quickly, without the upfront capital expense. This allows enterprises, start-ups, small and medium-sized businesses, and customers in the public sector to access the building blocks they need to respond quickly to changing business requirements. This white paper provides you with an overview of the services of the AWS Cloud and introduces you to the services that make up the platform.

In 2006, Amazon Web Services (AWS) began offering IT infrastructure services to businesses as web services—now commonly known as cloud computing. One of the key benefits of cloud computing is the opportunity to replace upfront capital infrastructure expenses with low variable costs that scale with your business. With the cloud, businesses no longer need to plan for and procure servers and other IT infrastructure weeks or months in advance. Instead, they can instantly spin up hundreds or thousands of servers in minutes and deliver results faster.

Today, AWS provides a highly reliable, scalable, low-cost infrastructure platform in the cloud that powers hundreds of thousands of businesses in 190 countries around the world.

7.1.1 Amazon EC2 instances

Cloud computing is the on-demand delivery of compute power, database, storage, applications, and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing. Whether you are running applications that share photos to millions of mobile users or you're supporting the critical operations of your business, a cloud services platform provides rapid access to flexible and low-cost IT resources. With cloud computing, you don't need to make large upfront investments in hardware and spend a lot of time on the heavy lifting of managing that hardware. Instead, you can provision exactly the right type and size of computing resources you need to power your newest bright idea or operate your IT department. You can access as many resources as you need, almost instantly, and only pay for what you use.

Cloud computing provides a simple way to access servers, storage, databases and a broad set of application services over the Internet. A cloud services platform such as Amazon Web Services owns and maintains the network-connected hardware required for these application services, while you provision and use what you need via a web application.

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. The simple web interface of Amazon EC2 allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances (called Amazon EC2 instances) to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers and system administrators the tools to build failure resilient applications and isolate themselves from common failure scenarios.

Instance Types:

- On-Demand Instances—With On-Demand instances, you pay for compute capacity by the hour with no long-term commitments. You can increase or decrease your compute capacity depending on the demands of your application, and only pay the specified hourly rate for the instances you use. The use of On-Demand instances frees you from the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly large fixed costs into much smaller variable costs. On-Demand instances also remove the need to buy "safety net" capacity to handle periodic traffic spikes.

- Reserved Instances—Reserved Instances provide you with a significant discount (up to 75%) compared to On-Demand instance pricing. You have the flexibility to change families, operating system types, and tenancies while benefitting from Reserved Instance pricing when you use Convertible Reserved Instances.

- Spot Instances—Spot Instances are available at up to a 90% discount compared to On-Demand prices and let you take advantage of unused Amazon EC2 capacity in the AWS Cloud. You can significantly reduce the cost of running your applications, grow your application's compute capacity and throughput for the same budget, and enable new types of cloud computing applications.
  When you launch an instance, the instance type that is specified determines the hardware of the host computer used for your instance.
  Each instance type offers different compute, memory and storage capabilities and is grouper in an instance family based on these capabilities. Select an instance type based on the requirements of the application or software that you plan to run on your instance.
  Amazon EC2 provides each instance with a consistent and predictable amount of CPU capacity, regardless of its underlying hardware. EC2 dedicates some resources of the host computer, such as CPU, memory and instance storage, It shares other resources of the host computer, such as network and disk subsystem, among instances.
  The minimum performance from a shared resource provided by each instance can be higher or lower.
  Ec2 provides a wide selection of instance types optimized for different use cases.
  There is an instance type available for free trial. For this one is not possible to choose a dedicated GPU hardware; P and G instance types are specific for Machine Learning application.[18]
  Each instance family (P and G) includes instance types, and each instance type includes instances with different sizes. Each instance has a certain vCPU count, GPU memory, system memory, GPUs per instance and network bandwidth. A full list of all available options is shown in the figure 7.2:

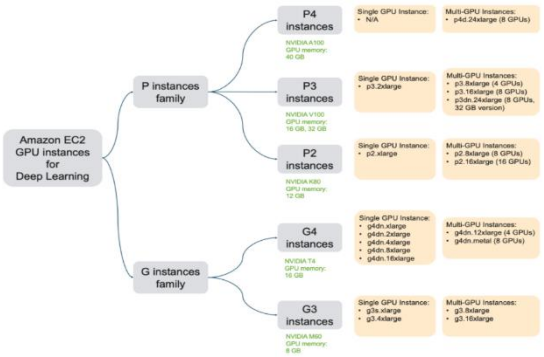


Figure 7.2

The highest performing deep learning training instance (P4) provides access to NVIDIA V100 GPUs based on NVIDIA Ampere architecture. You can also launch a multi GPUs per instance with 8 A100 GPUs with 40 GB of GPU memory per GPU, 96 vCPU and 400 Gbps network bandwidth for record setting training performance. P2 instances give you access to NVIDIA K80 GPUs based on the NVIDIA Kepler architecture. Kepler's architecture is a few generations old, therefore they're not the fastest GPUs around. They do have some specific features such as full precision support that makes them attractive and cost-effective for high performance computing (HPC) workloads that rely on the extra precision. P2 instances can be of 3 different sizes:

- P2.xlarge (1 GPU)
- P2.8xlarge (8 GPUs)
- P2.16xlarge (16 GPUs)

The NVIDIA K80 is an interesting GPU. A single NVIDIa K80 is actually two GPU on a physical board, which NVIDIA calls dual-GPU design. Launching an instance of p2.xlarge consists in the choice of what of these two GPU must be on K80 board.
P2 instance features:

➔ GPU generation: NVIDIa Kepler
➔ Supported precision types: FP64, FP32
➔ GPU memory: 12 GB
➔ GPU interconnect: PCIe

Prior to the launch of amazon EC2 G4 instances, P2 were a recommended cost-effective deep learning training instance type. Since the launch of G4 instances, P2 continues to be cost-effective, but without several features such as support for mixed-precision training and reduced precision inference improved with NVIDIA Ampere architecture.
Without optimized software, there is a risk of under-utilize of the hardware resources provided. A service highly recommended is AWS Deep Learning AMIs. By using it what is done is a test end to end giving the best performance.

AWS Deep Learning AMI: the DLAMI is one-stop shop for deep learning in the cloud. This customized machine instance is available in most Amazon EC2 region for a variety of instance types, from a small CPU-only instance to the latest high-powered multi-GPU instances. It comes preconfigured with NVIDIA CUDA and NVIDIA cuDNN, as well as the latest releases of the most popular deep learning frameworks.

8. Test phase: 2D pose estimation

Hardware:

- e.DO robot
- Camera (generic camera, no depth acquisition are needed)
- GPU: Intel HD Graphic 630

Software:

- OS: Ubuntu 18.04
- Environment: ROS Kinetic
- Programming language: Python (3.6 or higher)
- Main Git libraries:
  - SDKs: pyedo, realsense
  - Solver: tinyIK, IKPy
  - Image processing: openCV
  - CNN/Deep Learning elaborations: OpenPose

8.1 Test expected result

The aim of this project is to implement a robotic system able to perform simple task, as pick – place. The idea is to program this kind of actions without traditional programming approach.

Learning from demonstration is the approach that make possible to program a robot, without any programming skills. Starting from this idea, the goal of the thesis is to produce a robot that can be configured by recognizing objects or human pose, with the ability to move following its perception. In this phase the goal is to reproduce a 2D movement.

8.2 Pose estimation from detected key points

The pose estimation was developed starting from a 2D estimation pose: for this goal could be used RealSense acquisition or pre-stored images. Third coordinate is not considered in this phase.

Starting from 2D pose, the final development for 3D movement was done; in this case the RealSense acquisition is needed because depth acquisition is fundamental to perceive the depth information. These files give the information about third coordinate of the space necessary for the implementation of 3D movement. Moreover, it makes more accurate the pose estimation and as a consequence the pose imitation.

In order to perform a bi-dimensional estimation some analytical computation must be considered: the acquisition system reference frame is not corresponding to the human reference

frame or the robot reference frame. The transformation matrix used for moving the coordinates from the acquisition system RF to the human one are the following:
Rotational matrix (body→camera)

$$R_{body\rightarrow cam} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

Translation vector (body→camera)

$$t_{body\rightarrow cam} = \begin{bmatrix} n_x \\ ny \\ 0 \end{bmatrix}$$

Transformation matrix (body→camera)

$$T_{body\rightarrow cam} = \begin{bmatrix} 1 & 0 & 0 & n_x \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & n_y \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

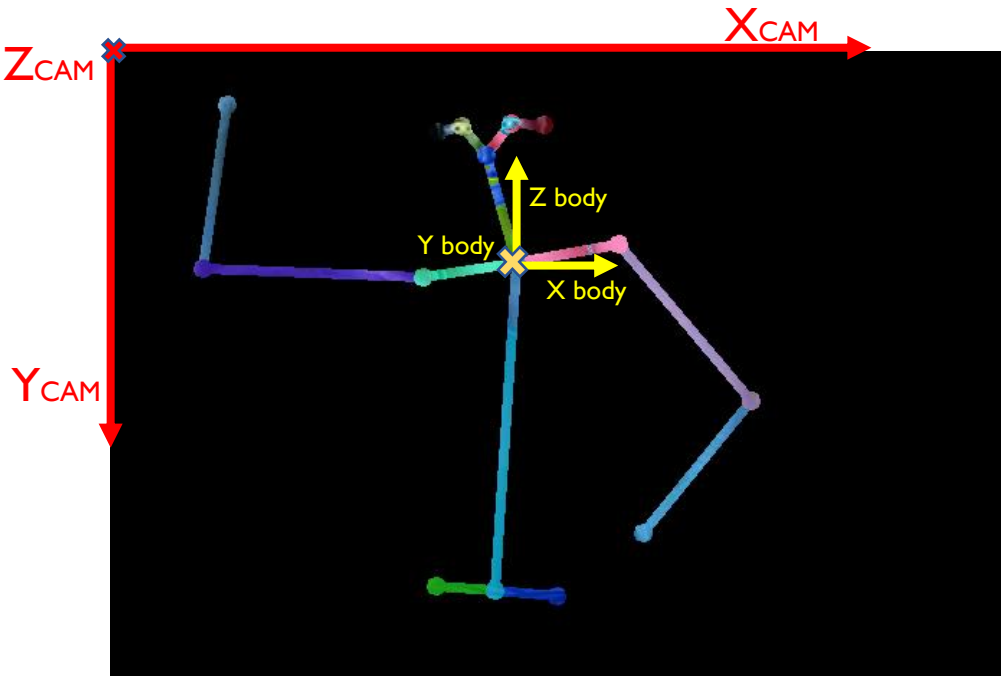


Figure 8.1: reference frames considered

8.3 Joint models computation

- J2 computation

In order to compute the joint position for joint 2, basic inverse kinematic solver is used: tinyIK. It is a simple solver that use a model created by the user, based on the specified rotational axis and length for each joint. The first joint specified is assumed attached at the base of the chain.

Using tinyIK the model of the first joint is defined as follows:

- Model = tinyIK.Actuator(['y', [0, 0, n_s]) where n_s = norm(neck-shoulder)

- Target position = T * (s_x, s_y, 0) **camera** rf (pixels) = (s_x, s_y, 0) **body** rf (pixels)

- Output angle = joint position to reach the shoulder detected point = 'j2'

This last computation get out the angle joint position to be set for the robot movement. Through using pyedo SDK the joint position is communicated and the robot performs the movement with the aim to reach this joint position.

- *J3 computation*

In order to compute the joint position for joint 3 a rotation of body reference system is taken into account (j2 around y). First, a Transformation matrix must be considered, because of the previous joint rotation.

In fact, pyedo SDK considers the configuration angle as relative rotation: each joint position is referred to the previous joint rotation:

- Transformation matrix = $\begin{bmatrix} cos(J2) & 0 & -sin(J2) & 0 \\ 0 & 1 & 0 & 0 \\ sin(J2) & 0 & cos(J2) & n_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Model = tinyIK.Actuator(['y', [0, 0, s_e]) where s_e = norm(shoulder-elbow)

- Target position = T * T1 * (e_x, e_y, 0) **camera** rf = T1 * (s_x, s_y, 0) **body** rf

- Output angle = joint position to reach the elbow detected point = j3

J5 computation

For computing the joint position for third joint previous rotation are taken into account (j2 and j3). As first transformation matrix is computed:

- Transformation matrix = $\begin{bmatrix} cos(J2+J3) & 0 & -sin(J2+J3) & n_s * sin(J3) \\ 0 & 1 & 0 & 0 \\ sin(J2+J3) & 0 & cos(J2+J3) & -s_e - n_s * cos(J3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Model = tinyIK.Actuator(['y', [0, 0, e_w]) where e_w = norm(elbow-wrist)

- Target position = T2 * T * (w_x, w_y, 0) **camera** rf = T2 * (e_x, e_y, 0) **body** rf

- Output angle = joint position to reach the wrist detected point = j5

An example of the output of the script used for the angle computation is reported below:

```
# 2D
# neck, shoulder, elbow, wrist (CAMERA rf)
n = np.array([652.22235,  497.15414, 0])
s = np.array([507.69153, 497.20465 , 0])
e = np.array([347.42737 ,434.09454, 0])
w = np.array([189.74858, 323.7846 , 0])
print("neck CAM: ", n)
print("shoulder CAM: ", s)
print("elbow CAM: ", e)
print("wrist CAM: ", w)

# neck, shoulder, elbow e wrist (CAMERA rf) homogeneous coordinates
nq = np.array([n[0],n[1],n[2], 1])
sq = np.array([s[0],s[1],s[2], 1])
eq = np.array([e[0],e[1],e[2], 1])
wq = np.array([w[0],w[1],w[2], 1])

print("n in omogenee CAM: ",nq)
print("s in omogenee CAM: ",sq)
print("e in omogenee CAM: ",eq)
print("w in omogenee CAM: ",wq)

nq = [n[0],n[1],0, 1]
sq = [s[0],s[1],0, 1]
eq = [e[0],e[1],0, 1]
wq = [w[0],w[1],0, 1]

print("n in omogenee CAM: ",nq)
print("s in omogenee CAM: ",sq)
print("e in omogenee CAM: ",eq)
print("w in omogenee CAM: ",wq)


# Rotation of 90 degree about x axis
#print("Matrice di Rotazione R(x,+90) = \n", R)

# New origin coordinates ---> {xcamera(neck), ycamera(neck)}

T = np.array([[1,0,0,-n[0]], [0,0,1,0], [0,-1,0,n[1]],[0,0,0,1]])

# Roto-traslation
print("Matrice di Roto-Traslazione T =\n ", T)

nr0 = np.dot(T,nq)
sr0 = np.dot(T,sq)
er0 = np.dot(T,eq)
wr0 = np.dot(T,wq)

print("Origin coordinates BODY: \n", nr0[0:3])
print("Shoulder coordinates BODY: \n", sr0[0:3])
print("Elbow coordinates BODY: \n", er0[0:3])
print("Wrist coordinates BODY: \n", wr0[0:3])

ns = np.linalg.norm(nr0-sr0)
m1 = tinyik.Actuator(['y', [0,0,ns]])
m1.ee = sr0[0:3]
a1 = m1.angles
```

```
T1=np.array([[cos(a1), 0, -sin(a1), 0],[0,1,0,0], [sin(a1), 0, cos(a1), -
ns], [0,0,0,1]])
print("origin model s-e: \n", np.round(np.dot(T1,sr0)))

se = np.linalg.norm(sr0-er0)
m2 = tinyik.Actuator(['y', [0,0,se]])
er1 = np.dot(T1,er0)
m2.ee = er1[0:3]
a2 = m2.angles



print(er0)
t = np.array([0,0,se])+[0,0,ns*cos(a2)]
print(t)

T2 = np.array([[cos(a2+a1), 0, -sin(a2+a1), ns*sin(a2)],[0,1,0,0],
[sin(a1+a2), 0, cos(a1+a2), -se-ns*cos(a2)], [0,0,0,1]])
print(T2)
print("origin model e-w: \n", np.round(np.dot(T2,er0)))

ew=np.linalg.norm(er0-wr0)
m3 = tinyik.Actuator(['y',[0,0,ew]])
ew2 = np.dot(T2,wr0)
m3.ee = ew2[0:3]
a3 = m3.angles


print("angle between z-ns: \n", np.rad2deg(a1))
print("angle between ns-se: \n", np.rad2deg(a2))
print("angle between se-ew: \n", np.rad2deg(a3))
```

The output skeleton obtained from the script is shown in figure 8.2:

```
Origin coordinates BODY:
 [0. 0. 0.]
Shoulder coordinates BODY:
 [-49.97377014   0.         -2.10424805]
Elbow coordinates BODY:
 [-95.38021469   0.        -80.19100952]
Wrist coordinates BODY:
 [ -80.23180008    0.        -140.93841553]
origin model s-e:
 [0. 0. 0. 1.]
[-95.38021469   0.        -80.19100952   1.        ]
[  0.           0.        117.26864126]
[[  -0.86447272    0.           0.50267973  -42.14319811]
 [   0.            1.           0.           0.        ]
 [   -0.50267973   0.          -0.86447272 -117.26864126]
 [   0.            0.           0.           1.        ]]
origin model e-w:
 [ 0.  0. -0.  1.]
angle between z-ns:
 [-92.41113233]
angle between ns-se:
 [-57.41141917]
angle between se-ew:
 [-44.17956623]
```

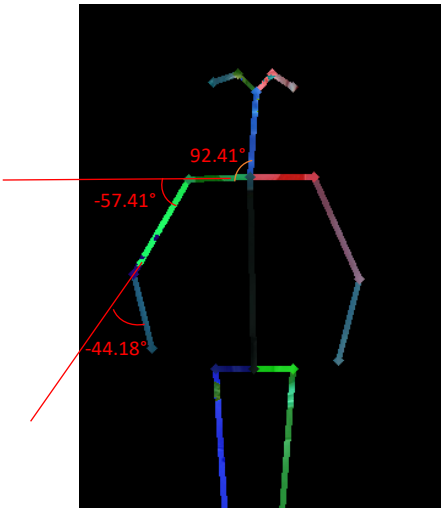


Figure 8.2: output skeleton

Considering the pixel coordinate information extracted by OpenPose, the angle between each joint is computed. Consequently, these values are sent to the robot in order to obtain the configuration demonstrated. The function used to communicate the values to the robot is the move_joint() set by pyedo SDK (chapter 4).

This function expects the value of the relative angular position starting from the first joint (absolute reference frame).

move_joint(ovr, j1, j2, j3, j4, j5, j6, j7)

where:

- ovr: speed of movement (% of maximum possible value)
- jn (n=1,…,6): rotating joints; same rotation axis (z_{robot}) for j2, j3, j5, same rotation axis (y_{robot}) for j1, j4, j6.
- j7: gripper opening

8.4 Test result: 2D pose imitation

The test consist of few steps:

1. Acquisition of the pose
2. Detection of key points
3. Skeleton and joint angles computation
4. Programming of the pose on the robot

On the left is reported the input images, used as demonstration of the pose. The skeleton computed after key points detection is overlapped on input image.On the right is reported the response of the robot to the pose demonstrated. After angle computation the robot is controlled in joint configuration through these parameters:

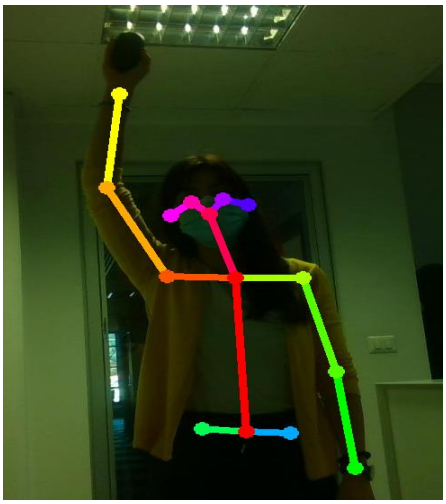


Figure8.3: Pose demonstration

Figure 8.4:: Robot imitation

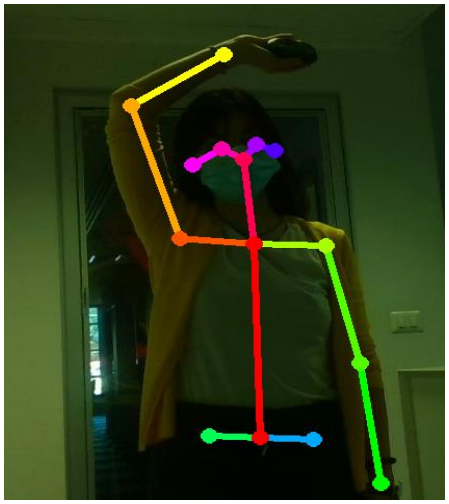


Figure 8.5: Pose demonstration

Figure 8.6: Robot imitation

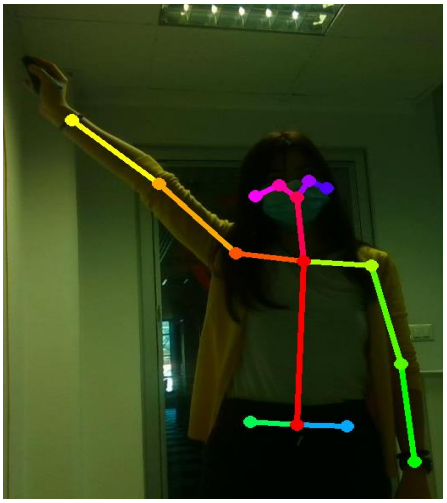


Figure 8.7: Pose demonstration

Figure 8.8: Robot imitation

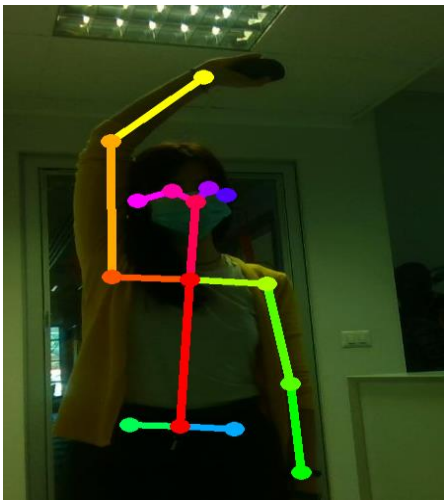


Figure 8.9: Pose demonstration

Figure 8.10: Robot imitation

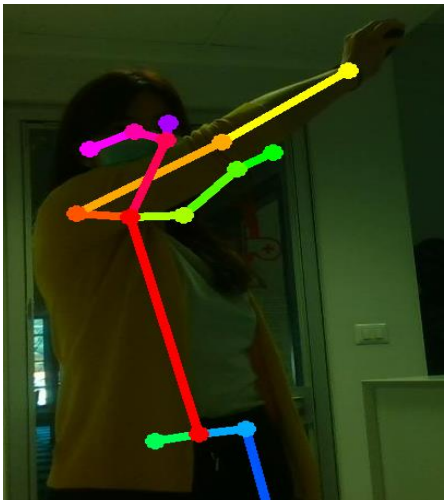


Figure 8.11: Pose demonstration

Figure 8.12: Robot imitation

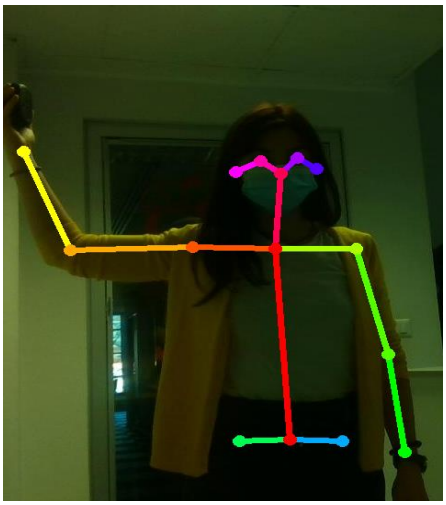


Figure 8.13: Pose demonstration

Figure 8.14: Robot imitation

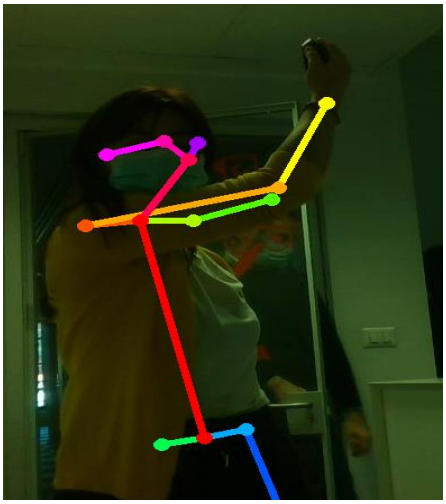


Figure 8.15: Pose demonstration

Figure 8.16: Robot imitation

The test have reported some distinctive characteristics, useful to study a solution for next steps.

In the tests performed:

Elaboration frequency is approximately 1/50 (time for image processing and key point recognition).

Factors influencing processing time:

1) Partial body view (torso and face only).

2) Weak contrast between body and background

3) Objects other than the object of interest present in the acquisition environment

Taking into account the results of these tests and considering the intrinsic characteristics of the system, the next step is to perform a demonstration in 3-dimensions in order to consider the total chain.

9. Test phase: 3D pose estimation

Starting from 2D implementation, the aim is to extend the movement imitation to 3D case. The main variation from 2D case is the input image: at this point the acquisition method is mandatory to be a depth camera. Depth sensibility is needed to integrate the third coordinate.

9.1 Pose estimation from detected key points

The robot chain is characterized by 6 joints. Each joint is rotational and linked to the next joint: each angle position is relative to the previous joint angular position. Computation must be referred to the previous joint orientation.

Reference systems

Body:

The origin of the body reference frame is considered on neck (position keypoint detected by openpose)

- z axis direction: UP
- y axis direction: BACK
- x axis direction: RIGHT
-

Robot:

The origin of the robot reference frame is considered on the fixed base joint (beginning of joint 1)

- z axis direction: UP
- y axis direction: BACK
- x axis direction: RIGHT

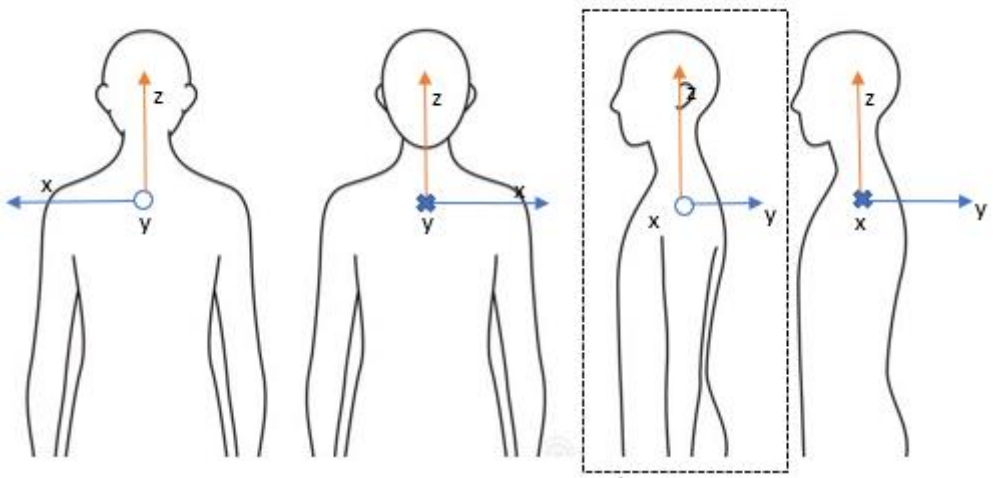


Figure 9.1: body reference system from different point of views. The selected one is analogue to the robot reference system

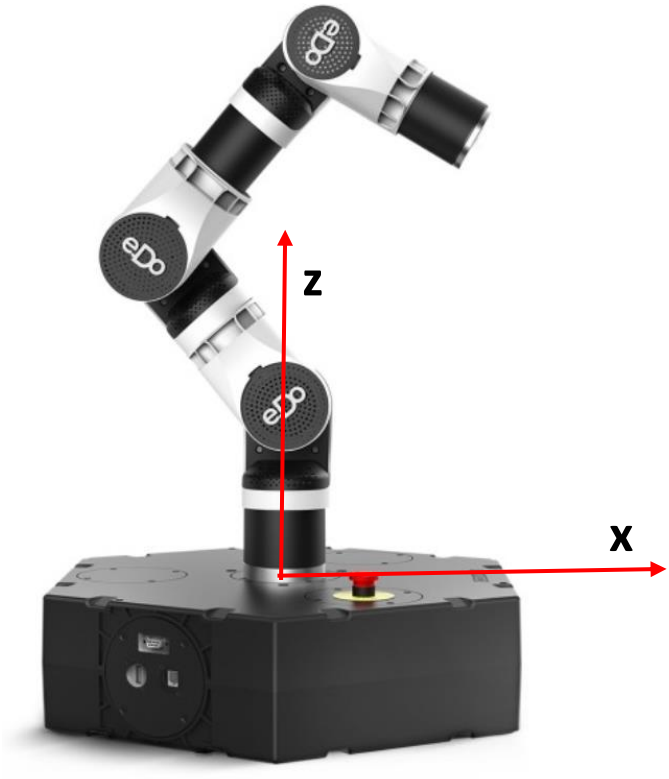


Figure 9.2: robot reference system

Considering the reference shown in figure, some assumption can be done to put in evidence the analogy between the human chain and the robot chain. In the following table are explained the association of each human joint (neck-shoulder, shoulder-elbow, elbow-wrist) to the robot ones. The human links are three, but able to performs rotation about more than one rotation axis. By this assumption is possible to refer one human joint to more than one robot joint because of the movements of the human one can be reproduced in robot thanks to the composition of two joints rotation.

Joint	Rotation axis	Movement description	Origin
J1	z	n-s (piano xy)	Neck (n)
J2	y	n-s (piano xz)	Neck (n)
J3	y	s-e (piano xz)	Shoulder (s)
J4	z	s-e (piano xy)	Shoulder (s)
J5	y	e-w (piano xz)	Elbow (e)
J6	z	e-w (piano xy)	Elbow (e)

(nx, ny, nz) = neck coordinates = n

(sx, sy, sz) = shoulder coordinates = s

(ex, ey, ez) = elbow coordinates = e

(wx, wy, wz) = wrist coordinates = w

The computation of each joint position:

- j2, j3, j5 = rotation in xz plane, about y axis.
  x and z coordinates are estimated from RGB image.

- j1, j4, j6 = rotation in xy plane, about z axis.
  x coordinate is estimated from RGB image;
  y coordinate is estimated from depth image.

9.2 Image processing in depth acquisition

Depth RGB camera get out two kinds of acquisition; these two outputs have different dimension and resolution. Following the logic of 2D movement the RGB acquisition is used to detect the key points (neck-shoulder-wrist) coordinates. These coordinates are expressed in pixels.

Depth acquisition is used to estimate the 3rd coordinate. The information about it is linked to RGB scale expressing the depth vision of the camera. Depth is a color mapped information depending on the setting RGB scale.

The RGB values are not comparable to the pixel measurement unit, some trials have been performed but the relevance of the plane 2D movement results greater than the depth perception and response.

For this reason, the coordinates had to be translated in a univocal measurement system, in order to obtain the same results independently on the perception of the camera.

Considering a pixel, defined by

- 3 coordinates: two coordinates used to define 2D position and one is used to express the distance from acquisition reference frame.
- 1 RGB value: is a 1x3 vector, with elements linked to the intensity of Red Green Blue color to be associated.

Assuming as RF the body, each pixel is considered referred to the neck key point (considered at zero position).

Each key point (shoulder, elbow, wrist) coordinate is linked to the neck relative coordinate as scalar distance. Now each joint movement is considered, so the distances are viewed on two different planes to consider also movements on 3rd direction.

As in the following figure (33), robot joints and human arm joints are associated by analogy. The human arm reference frame is referring to the robot reference frame. Starting from these assumptions, it is possible to compute the movement for the robot starting from the human movement.

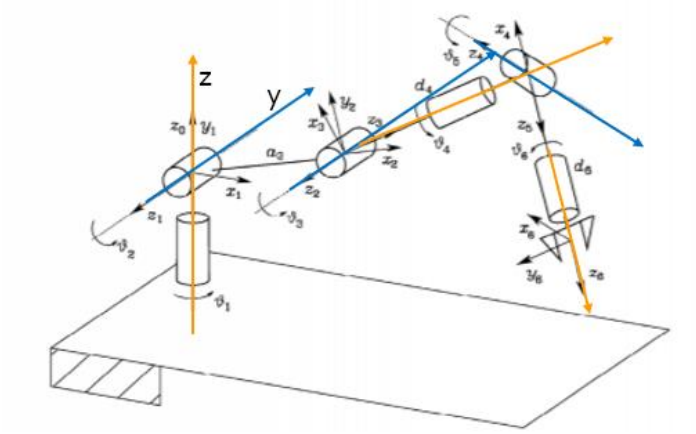


Figure 9.3

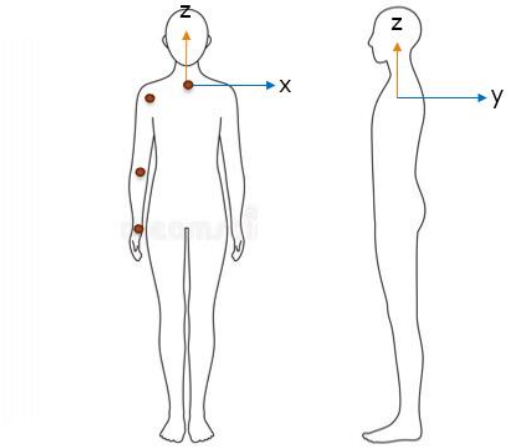


Figure 9.4

Assuming the robot configuration, the joint **analogy** on the human arm is:

j1 = torso

j2 = neck-shoulder

j3 = shoulder-elbow moving around y

j4 = shoulder-elbow moving around z

j5 = elbow-wrist moving around y

j6 = elbow-wrist moving around z

8.3.3 Inverse kinematics solution

In order to compute the angle configuration for the robot the first analysis is about the movement in two dimension. Once this analysis gets the resulting angle the same procedure is applied on the other plan of movement. The joints mainly included in this phase are the first 5 joints. The sixth joint is not fundamental for the imitation of the pose because its main role is giving the direction of arrival to the gripper (last joint).

In order to compute the joint position for each joint, a basic inverse kinematic solver (tinyIK) is used. It is a simple solver where the model can be created specifying the rotation axis and the lenght of the rotating joint. This phase is the same of the two-dimension experiment. The first joint specified is assumed attached at the base of the chain.

Some assumptions to have the arm pointing up as 'zero' configuration:

Torso rotation = ***j1 = 0***

Neck-shoulder rotation around y = ***j2 = 0*** *

Shoulder-elbow rotation around y = ***j3 = angle3 - j2***

*(if not compensated the joint j2 goes always to -90° with respect to vertical axis)

The model for each joint is expressed in the robot reference system, relative to the rotation of previous joint.

The joint1 is referring to the absolute robot reference frame. The joint from 2 to 6 are referring to the RF rotated about rotation axis and of the angle of the previous joint/joints. The angle of the joints having the same rotation axis is summed together with the others. In this sense the total movement is the composition of the movement on two planes (two different rotation axis). The joints j2, j3 and j5 have rotation axis 'y_{robot}'. The joints j1, j4, j6 have rotation axis 'z_{robot}'. The change of reference frame is expressed by transformation matrices that consider the rotation and the length of each joint. Steps to compute each angle final position are:

1. Projection of the joint in new rotated RF
2. Consideration of the Target position as coincident with the origin of the next joint (end point of the actual joint).
3.

9.3 Robot joint models analogy with human joints

J1:

```
#model J1
ns_norm = np.linalg.norm(n_cm-s_cm)
ns = [0,0,ns_norm,1]
print("starting pose: \n", ns)
m1 = tinyik.Actuator(['z', ns[0:3]])
target1 = np.array([s_cm[0],s_cm[1],0])
print("\n target point1 xz: \n", target1)
m1.ee = target1
```

```
a1 = m1.angles
print("\n J1 =", np.rad2deg(a1))
```

J2:

```
#model J2
ns_rot = np.dot(R1,ns)
print("\n starting pose after J2: \n", ns_rot)
m2 = tinyik.Actuator(['y', ns_rot[0:3]])
target2 = np.array([s_cm[0],0,s_cm[2]])
print("\n target point2 xy: \n", target2)
m2.ee = target2
a2 = m2.angles
print("\n J2 =", np.rad2deg(a2))
```

J3:

```
#model J3
es_norm = np.linalg.norm(e_cm-s_cm)
es = [0,0,es_norm,1]
e_srf = np.dot(T12,e_cm)
m3 = tinyik.Actuator(['y', es[0:3]])
target3 = np.array([e_srf[0], 0, e_srf[2]])
print("target point 3 xz: \n", target3)
m3.ee = target3
a3 = m3.angles
print("\n J3 = ", np.rad2deg(a3))
```

J4:

```
es_rot = np.dot(R3,[0,0,e_srf[2],1])
print("\n starting pose after J3: \n", es_rot)
m4 = tinyik.Actuator(['z', es_rot[0:3]])
target4 = np.dot(R3,np.array([e_srf[0],e_srf[1],0,1]))
print("\n target point 4 xy: \n", target4)
m4.ee = target4[0:3]
a4 = m4.angles
print("\n J4 = ", np.rad2deg(a4))
```

J5:

```
#model J5
ew_norm = np.linalg.norm(e_cm-w_cm)
ew = [0,0,ew_norm,1]
w_erf = np.dot(T12_34,w_cm)
m5 = tinyik.Actuator(['y', ew[0:3]])
target5 = np.array([w_erf[0], 0, w_erf[2]])
print("\n target point 5 xz: \n", target5)
m5.ee = target5
a5 = m5.angles
print("\n J5 = ", np.rad2deg(a5))
```

J6:

```
#model j6
ew_rot = np.dot(R5,[0,0,w_erf[2],1])
print("\n starting pose after J5: \n", ew_rot)
m6 = tinyik.Actuator(['z', ew_rot[0:3]])
target6 = np.dot(R5, np.array([w_erf[0],w_erf[1],0,1]))
print("\n target point 6 xy: \n", target6)
m6.ee = target6[0:3]
a6 = m6.angles
print("\n J6 = ", np.rad2deg(a6))
```

Chain structure is depending on each rotation acquired/performed. Two chains are mainly considered because neck-shouder are the starting chain and are considered 'fixed': relative movements between neck and shoulder are negligible and for this reason not considered. Then, the first relative movement is referring to the neck-shoulder initial position and represents the shoulder-elbow chain:

```
#rotation matrix J2
R2 = np.array([[cos(a2), 0, sin(a2), 0], [0, 1, 0, 0], [-sin(a2), 0,
cos(a2), 0], [0, 0, 0, 1]])

# next joints
Rt0 = np.dot(R1,R2)
R12 = np.transpose(Rt0)
t12 = -np.dot(R12,s_cm)
T12 = np.array([[R12[0,0], R12[0,1], R12[0,2], t12[0]],[R12[1,0], R12[1,1],
R12[1,2], t12[1]], [R12[2,0], R12[2,1], R12[2,2], t12[2]], [0, 0, 0, 1]])
```

The second projection to be considered is representing elbow-wrist chain, starting from the previous chain:

```
#rotation J3
R3 = np.array([[cos(a3), 0, sin(a3), 0], [0, 1, 0, 0], [-sin(a3), 0,
cos(a3), 0], [0, 0, 0, 1]])
R4 = np.array([[cos(a4), -sin(a4), 0, 0], [sin(a4), cos(a4), 0, 0], [0, 0,
1, 0], [0, 0, 0, 1]])
Rt1 = np.dot(R3,R4)
Rt2 = np.dot(Rt0,Rt1)
R12_34 = np.transpose(Rt2)
t12_34 = -np.dot(R12_34, e_cm)
T12_34 = np.array([[R12_34[0,0], R12_34[0,1], R12_34[0,2],
t12_34[0]],[R12_34[1,0], R12_34[1,1], R12_34[1,2], t12_34[1]],
[R12_34[2,0], R12_34[2,1], R12_34[2,2], t12_34[2]], [0, 0, 0, 1]])
```

From all these assumptions is possible to compute and refer the robot joint configuration to the human pose. Human arm structure is not the same of the robot chain, but is possible to find some analogy. For this reason, the movement is not apparently the same. Anyway, the direction of each robot joint follows the associated arm joint in the correct way.

9.4 Test results

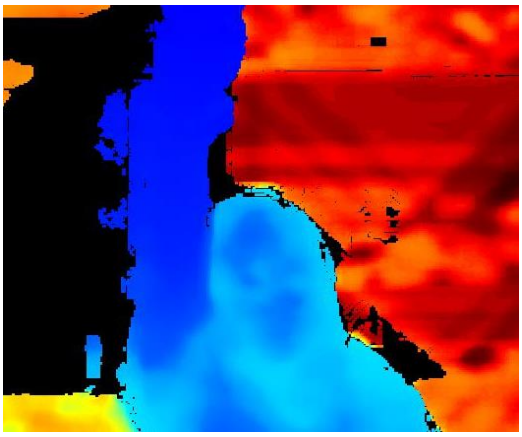
Following the experimental tests, some analyses are performed, following as much as possible the coherence with the shown movement: looking at the robot chain, it is evident that there is a

discrepancy between the joints of the robot and the joints of the human arm. The movements shown are different from the movement repeated by the robot, this is the reason why the gestures have been chosen in order to have a reasonable response easier to be recognized.

Considering the j2 (neck shoulder) angle configuration always zero, the initial position of the robot is identified in the straight arm pointing upwards.

The starting condition is the position on which are depending all the configurations: from this condition have been computed all the movements of the experimental tests.

1st pose: Forward point arm with no inclination on side direction



FRONT VIEW (piano xz)

J1 = 0

J2 = 1

J3 = -1

J4 = -87

J5 = 10

J6 = -1

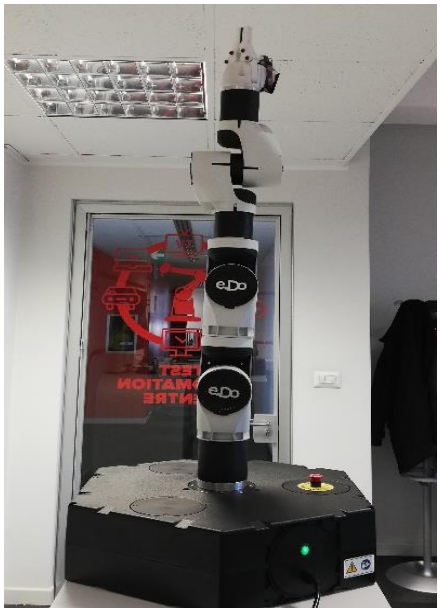
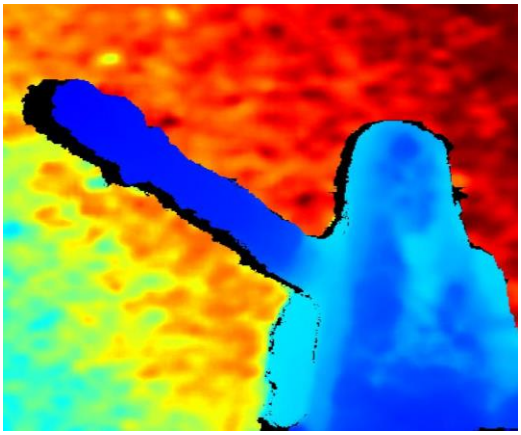


Figure 9.5

2nd pose: Forward-right point arm



FRONT VIEW (piano xz)

J1 = 0

J2 = 4

J3 = -60

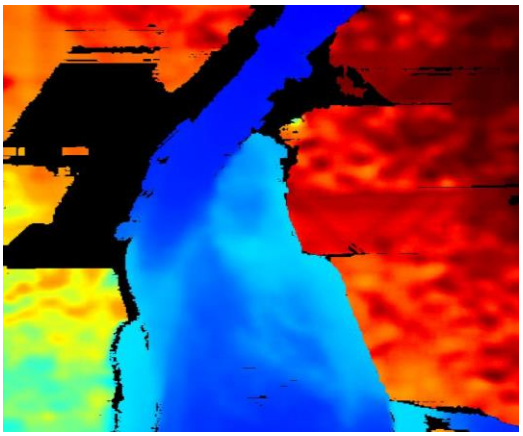
J4 = -59

J5 = 8

J6 = -7

Figure 9.6

3rd pose: Forward-left point arm



FRONT VIEW (piano xz)

J1 = 0

J2 = 5

J3 = 13

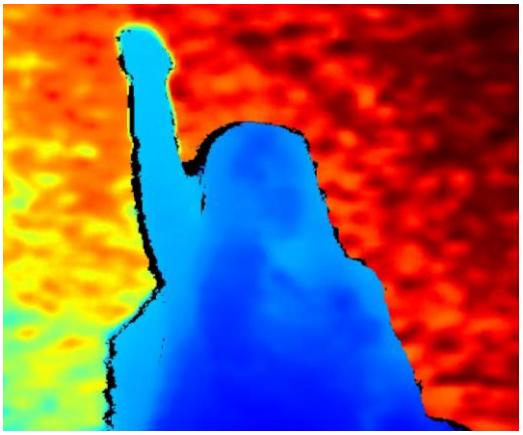
J4 = 81

J5 = -20

J6 = 55

Figure 9.7

4th pose: Backward point arm with no inclination on side direction



FRONT VIEW (piano xz)

J1 = 0

J2 = 10

J3 = -14

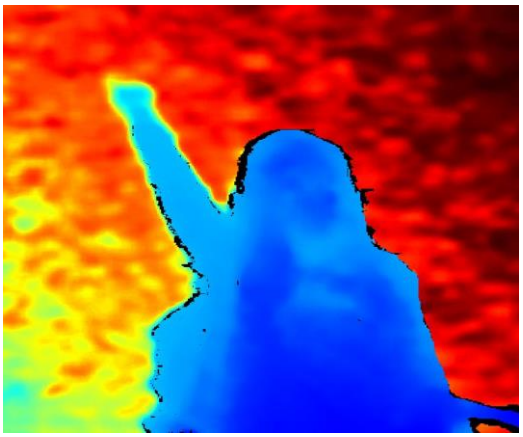
J4 = 42

J5 = -2

J6 = 77

Figure 9.8

5th pose: Backward-right point arm



FRONT VIEW (piano xz)

J1 = 0

J2 = 8

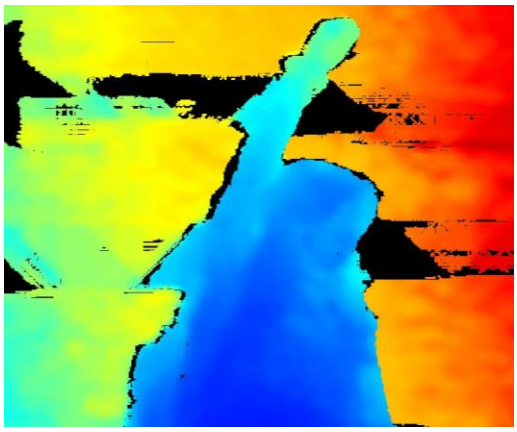
J3 = -30

J4 = 27

J5 = 20

Figure 9.9

6th pose: Backward-left point arm



FRONT VIEW (piano xz)

J1 = 0

J2 = 16

J3 = 7

J4 = 79

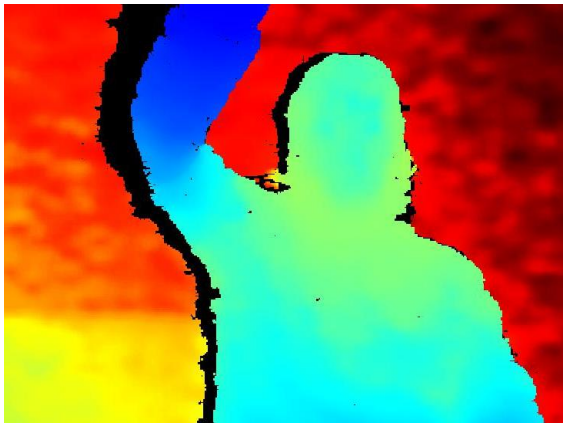
J5 = 16

Figure 9.10

From the tests performed, the movement in the yz plane is not as evident as that in the xz plane. The forward/backward displacement is perceived less during the acquisition than the right/left displacement; this could depend on the fact that, with the same number of centimeters, the value of the distance between RGB vectors is smaller than the value of the same distance expressed in pixels.

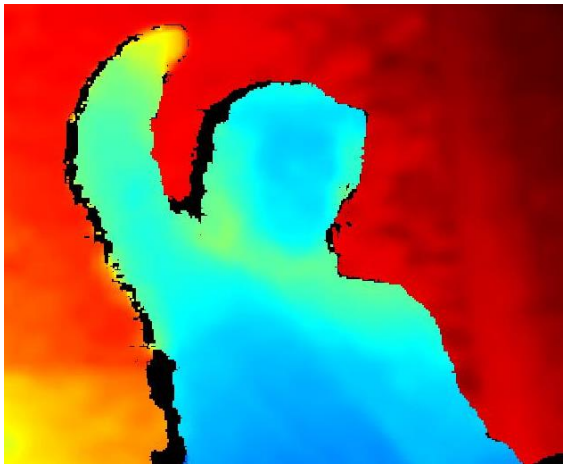
	J4 > 0	J4 < 0
J3 e/o J5 e/o (J3+J5) > 0	Backward - left	Forward - left
J3 e/o J5 e/o (J3+J5) < 0	Backward – right	Forward - right

As can be seen on the images below, if the human arm is pushed further forward the angular movement is greater (greater angles) because the differences in depth become more apparent to the camera (distances value RGB increases).



```
j1 su z:  [0.]
j2 su y:  [9.97771222]
j3 su y:  [-45.00000038]
j4 su z:  [-74.83910439]
j5 su y:  [61.23099944]
j6 su z:  [45.98523431]
```

Figure 9.11



```
j1 su z:  [0.]
j2 su y:  [4.23639444]
j3 su y:  [-30.69118152]
j4 su z:  [45.37784625]
j5 su y:  [73.91060281]
j6 su z:  [63.58060259]
```

Figure 9.12

Since the resolution of the 3D kinematics obtained with the coordinates expressed in pixels and depth-RGB value presents deficits in the interpretation of the depth-RGB values, it is necessary to report the three coordinates in the same reference system.

The transformation ratio pixel-meters is different from RGB meters, for this reason at the same meters the expression in pixel and the expression in RGB produce two different results. Since a coordinate must be expressed as an RGB value, and it is not possible to express it in the same reference system as the other two coordinates, we proceed by taking the 3 coordinates to the same unit of measurement (in this case meters).

The transformation of the points from pixels to meters is based on the calculation of parameters characteristic of the acquisition system, as well as the environment in which the test is performed.

- Intrinsic parameters

These ones link the acquisition system with the image plane within which the acquisition is expressed. It is possible to obtain an expression of the coordinates referred to the camera reference system through the use of these parameters:

Focal length:

$$f = 423{,}723\ (=f_x = f_y)$$

Center of image plane coordinates (expressed in pixel, resolution 840x840):

$$C_x = 420.892$$

$$C_y = 238.206$$

The intrinsic matrix results:

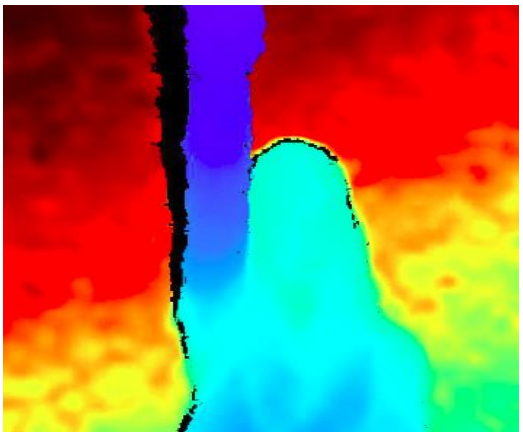
$$\begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

Considering point K as a key point extracted from OpenPose, expressed in the image reference system with units in pixels and depth expressed in RGB, the algorithm implemented to obtain the coordinates expressed in the body reference system with units in meters is as follows:

1. Extract scale factor (proportional to the distance of the image plane) of the camera (=k)
2. Extract key points from OpenPose
3. Compose intrinsic and extrinsic matrix
4. I transform the coordinates of the points extracted from OpenPose, through the use of matrices in step 4, from pixels to meters
5. Report the values in cm to the reference system neck, referring to the coordinates of the point "n" previously transformed into meters
6. I use 3D motion resolution algorithm to obtain the angles

9.5 Test result: key points coordinate expressed in cm

1st pose: Forward point arm with no inclination on side direction



J2 = 0

J3 = -4.5

J4 = -71.61

J5 = 18.86

J6 = -16.96

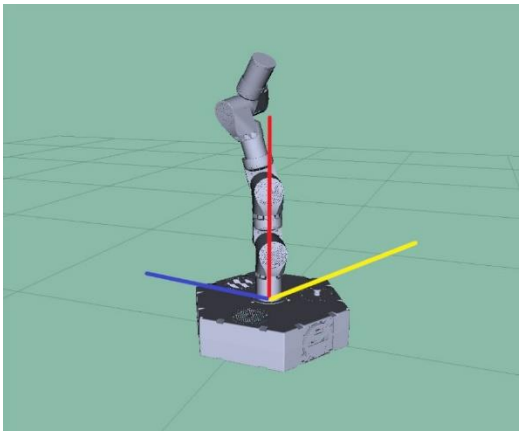
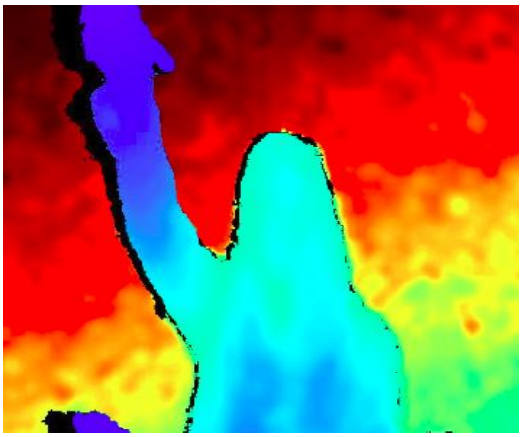


Figure 9.13

2nd pose: Forward - right point arm



J2 = 0

J3 = -45.53

J4 = -38.02

J5 = 38.71

J6 = -2.18

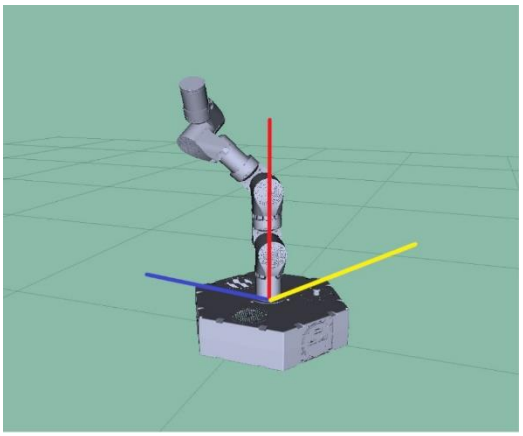
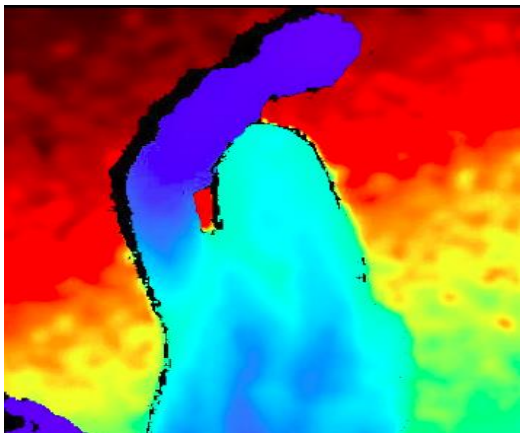


Figure 9.14

3rd pose: Forward – left point arm



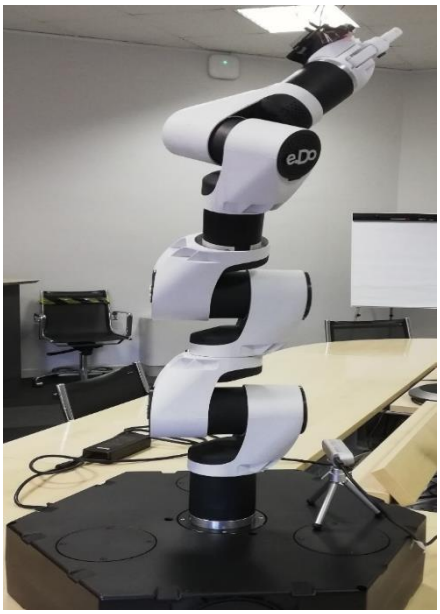
J2 = 0

J3 = -23.85

J4 = -44.38

J5 = 62.93

J6 = 48.35



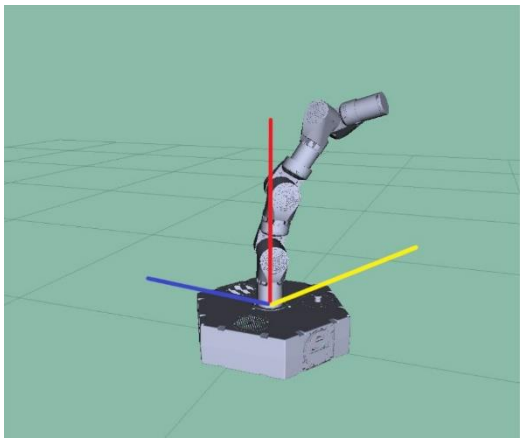
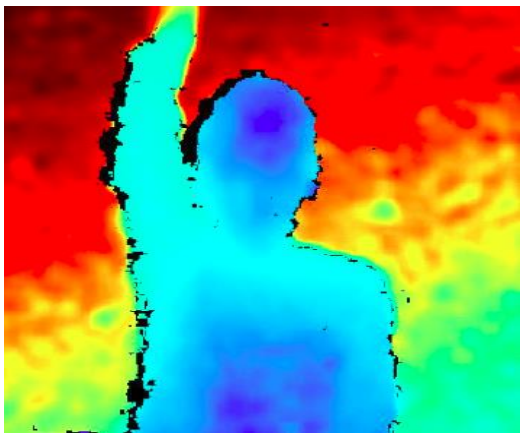


Figure 9.15

4th pose: Backward point arm with no inclination on side direction



J2 = 0

J3 = -10

J4 = 29.02

J5 = 37.08

J6 = 8.21

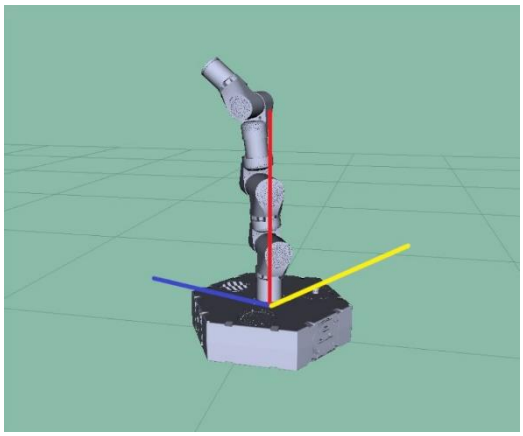
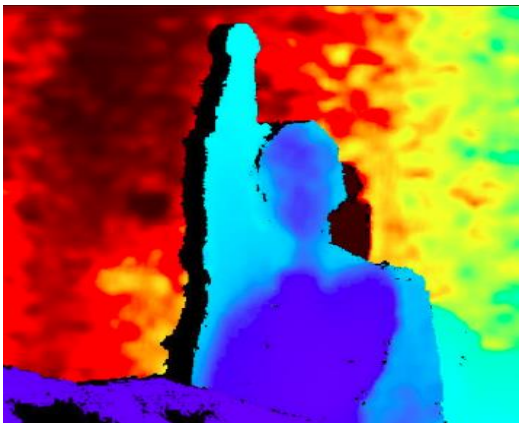


Figure 9.16

5th pose: Backward - right point arm



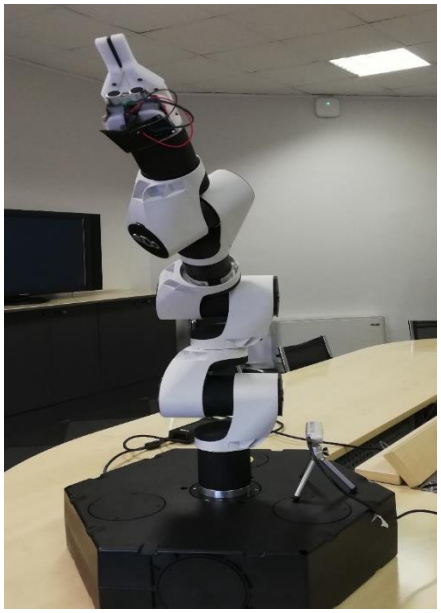
J2 = 0

J3 = -3

J4 = 38.17

J5 = 22.58

J6 = 27.96



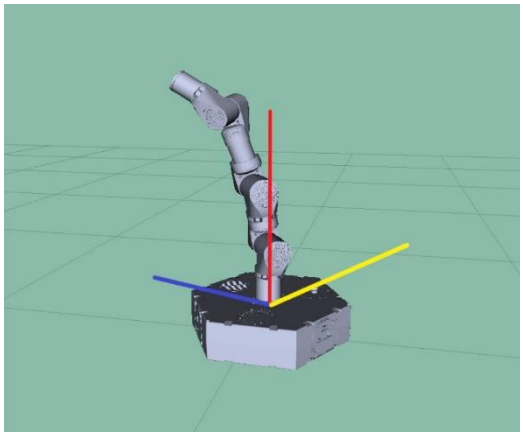
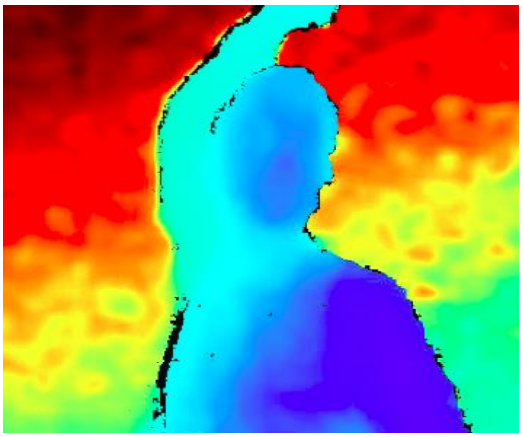


Figure 9.17

6th pose: Backward - left point arm



J2 = 0

J3 = -4

J4 = 49.95

J5 = 29.71

J6 = -66.85

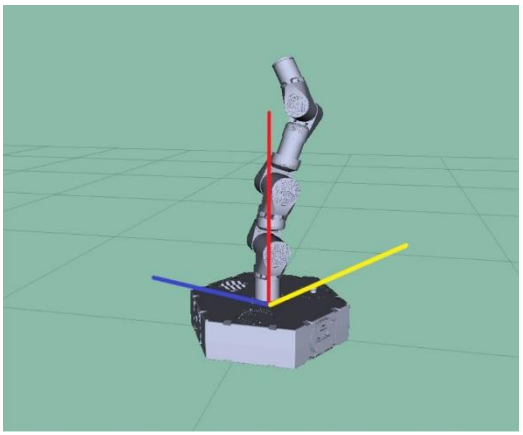


Figure 9.18

9.6 Space point detection

From depth acquisition is possible to implement another use case, with the aim to cover the case of collaborative tasks (i.e. exchanging objects) between user and robot. Considering this purpose, the depth acquisition is used to map space around the system.

For this goal becomes important the relative position between camera and robot because the point must be reached by the robot but is viewed at first by the camera. In the following figure, we can find the association between camera reference system and robot reference system:

To 'read' correctly the space around the system, the calibration process must be performed before starting the test.

The Robot and Camera reference system are shown in the following figure. According to these directions, some matrices are taken into account to consider the relative positions:

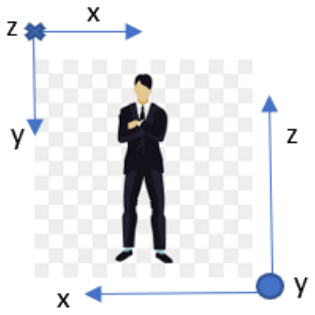


Figure 9.19: camera point of view

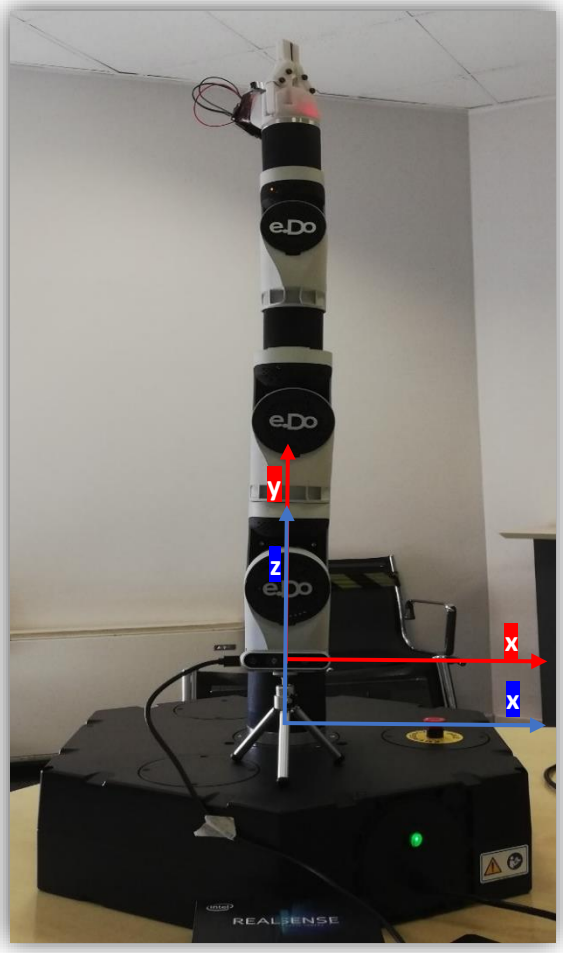


Figure 9.20: robot RF (y) camera RF (o)

Two rotation are performed in order to overlap the RFs. The two matrix are:

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The vectorial product (composition) of these two matrices gives the matrix R:

$$R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

9.6.1 Calibration

In order to express the coordinates in robot reference system:

- Origin of camera rf and robot rf must coincide: compensation of 15 cm z-coordinate*
- The URDF file does not consider the gripper length, the end effector rf originates on j6; in order to execute a picking action, the robot must reach the target with the gripper, so the origin of the end effector is translated of 6 cm on z-coordinate (a half of gripper length)

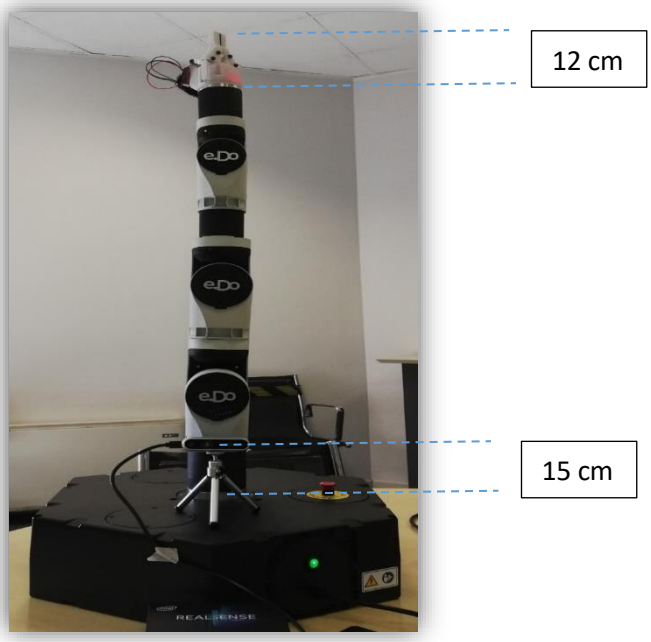


Figure 9.21

Precondition to test:

- User in front of the robot (and the camera)
- User shows the wrists as target to be reached

9.7 Test results: robot reaches target point in the space

1st test

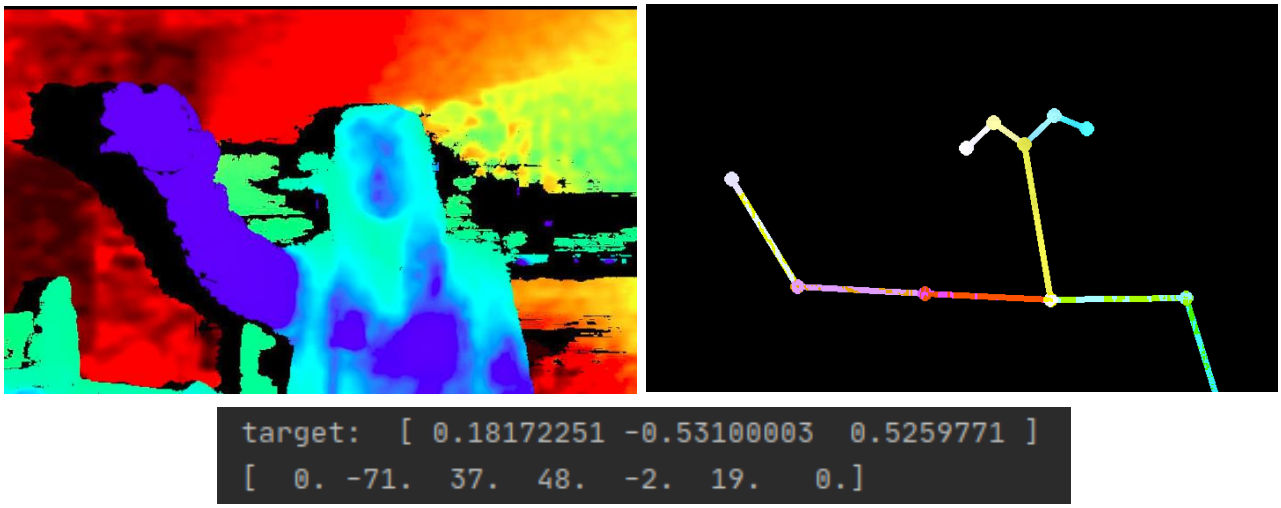
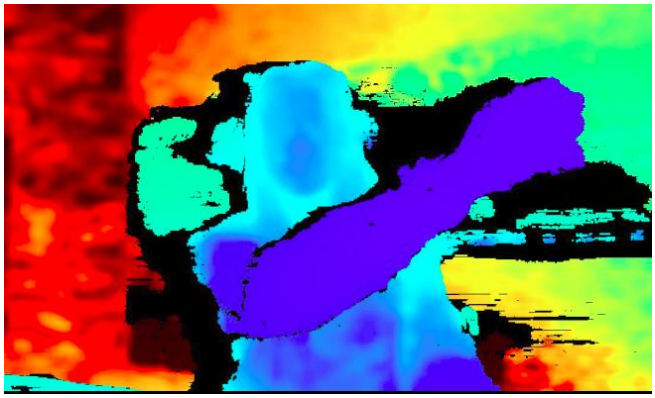
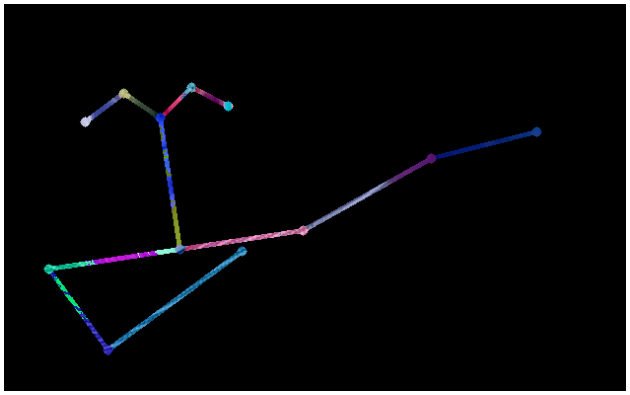


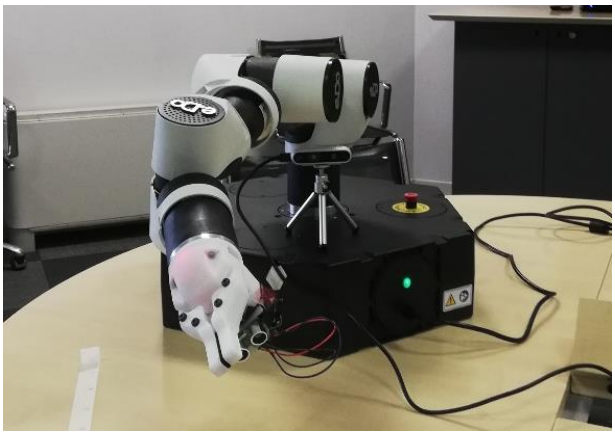
Figure 9.22: simulation

2nd test





```
target:  [-0.16189824 -0.60300003  0.3088541 ]
[  0.  67. -82. -19.  80. -28.   0.]
```



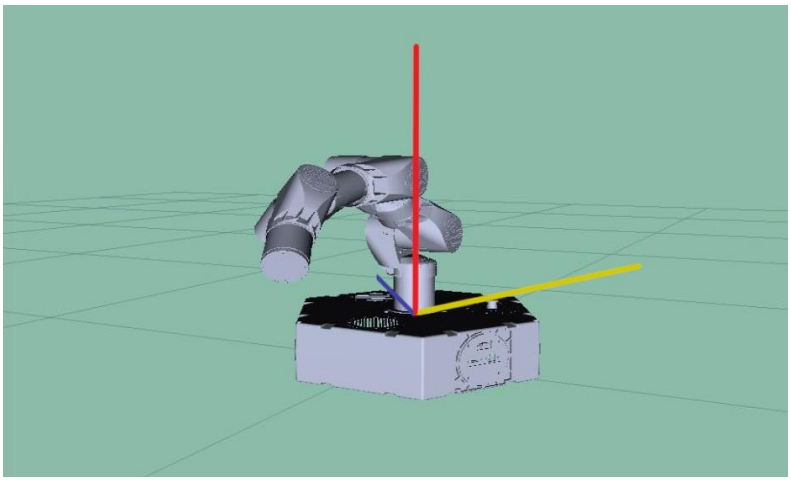


Figure 9.23: simulation

9.7 Performance analysis

The results obtained from the tests are consistent with those expected, but some adjustments could be made to improve performance. The analysis of the results showed room for improvement on two main fronts: versatility and responsiveness of the system.

In order to have the possibility to perform the same kind of tests in different environments, the calibration is the most influential phase. Specifically, it would be better to consider the translation between the robot reference system and the camera reference system (which was considered zero during the tests). As a result of this choice, it would be possible to obtain a system adaptable to different test environments, as well as the possibility of placing the camera in different positions from the one originally proposed.

This allows the system to improve in terms of accuracy (calculation of coordinates) and in terms of repeatability of the tests, as well as being validated in the most correct way.

Another aspect influencing performance was the latency time of the system. The processing of images has an average duration of 40 seconds, this translates into a delay in response on the system as a whole. The processing of acquired images is dependent on the characteristics of the machine used to process; fortunately, OpenPose can be used in two versions:

- On CPU

- On dedicated GPU (optimal performance)

The tests shown in the two previous chapters were carried out with the CPU version of OpenPose, with the specific hardware available. This led to an analysis, firstly based on the consistency of the robot in the imitation.

Following the tests carried out on CPU, the analysis has shifted to the aspect of temporal processing, in order to optimize performance albeit at low cost.

10. Amazon AWS and Web service integration

Considering the results obtained from the 3D tests shown in the last chapter, the response performance of the system was analyzed. In addition to the current results there are problems in terms of processing time due to the data processing hardware.

The system is in fact slow in response due to the lack of a GPU dedicated to the processing of images sent in input to Openpose. To obtain the optimal performance and as close as possible to a real-time point recognition is a Nvidia GPU, on which it is possible to use CUDA.

CUDA (stands for Compute Unified Device Architecture) is a hardware architecture for parallel processing created by NVIDIA. Through the CUDA development environment, software developers can write applications capable of running parallel computing on NVIDIA video card GPUs. Thanks to this architecture Openpose takes advantage of the computational power and outputs points in times similar to Real-Time.

This architecture compatibly with the Nvidia GPU has very high costs.

One of the objectives of the project is to obtain a robotic system that uses Deep Learning at low cost. Given these prerogatives it was decided to choose an educational manipulator and not industrial as it would have reported higher costs. The choice of the robot proved to be adaptable to the tasks developed.

On the other hand, the hardware needed to perform the image processing was not as performant as expected.

After the development of the tasks in terms of motion dynamics, we moved on to make an analysis of the time expenditure characterizing the tests carried out.

Considering the strong dependence of the processing on the architecture of the machine used in the processing, we moved on to an analysis of the hardware on which the system was implemented.

Specifically, the Nvidia GPU presents high costs that go against the objective of obtaining a low cost system.

In this regard, it was thought to take advantage of the virtual computing power made available by online services, in particular Amazon's AWS systems (Chapter 6).

These services offer the possibility to create a virtual machine with features chosen based on the user's needs. The difference with the typical virtual machines that can be used for free is that the hardware architecture is not based on the local one but is totally independent from the machine on which the service is requested.

10.1 Instance type

The requirements for obtaining the optimal development environment to use the Openpose library laid the groundwork on the choice of the specific instance among the various offered by the service. The instance chosen is the EC2 type: Amazon EC2 offers a wide range of instance types optimized to meet different use cases. Instance types include different combinations of

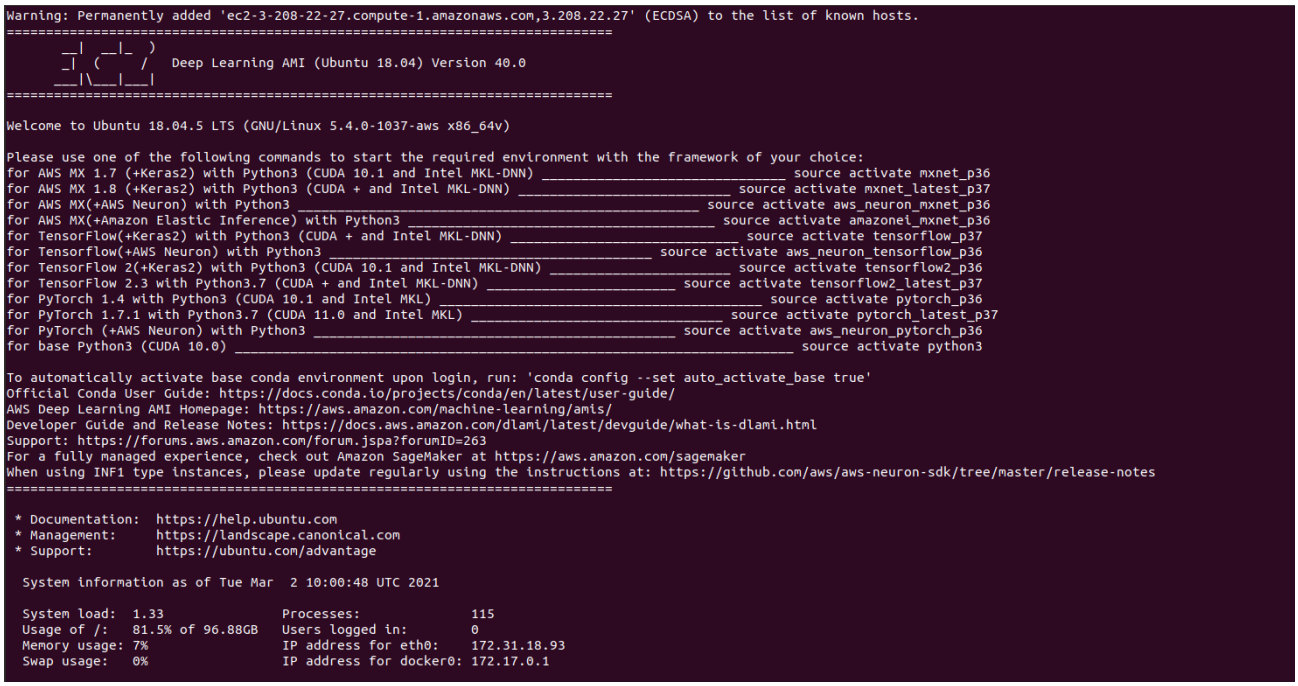
CPU, memory, storage, and network capabilities, giving you the flexibility to choose the appropriate combination of resources for your applications. Each instance type includes one or more instance sizes, allowing you to scale your resources according to workload requirements.

After choosing the instance type, the service offers several options in the field of Deep Learning. Several AMIs are distinguished: Deep Learning Base AMI is like an empty canvas for deep learning. It comes with everything you need up until the point of the installation of a particular framework. It will have your choice of CUDA versions. AWS Deep Learning Base AMI provides a foundational platform for deep learning on AWS EC2 with NVIDIA CUDA, cuDNN, NCCL, GPU Drivers, Intel MKL-DNN, Docker, NVIDIA-Docker and EFA support. This AMI is suitable for deploying your own custom deep learning environment at scale.

For example, for machine learning developers contributing to open source deep learning framework enhancements, the AWS Deep Learning Base AMI provides a foundation for installing your custom configurations and forked repositories to test out new framework features. You could also be a Machine Learning / AI startup with a highly specialized deep learning setup that needs a foundation to run on a cloud-scale infrastructure.

Another parameter required by Openpose is the number of cores per processor. It is in fact necessary to have 4 cores to obtain an optimal processing performance.

The main characteristic of the instance are reported below:



Following the creation of the machine with the suitable characteristics, it is necessary to proceed with the fundamental installations for reading the code executed so far on the local machine.

The Openpose library in particular is installed in GPU version (unlike the CPU version used locally).

10.2 Web service

In order to make the processing process callable from the virtual machine, a web service was implemented. A web service is a software system designed to support interoperability between different processors on the same network or in a distributed environment. This feature is obtained by associating to the application a software interface (described in an automatically processable format such as, e.g., Web Services Description Language) that exposes the Web Services Description Language, the Web Services Description Language) that exposes to the outside the associated service/s and using which other systems can interact with the same application activating the operations described in the interface (services or requests of remote procedures) through appropriate "messages" of request: such messages of request are included in an "envelope" (the most famous is SOAP), formatted according to the standard XML, encapsulated and transported through the protocols of the Web (usually HTTP), from which exactly the name Web service. In fact, the web service consists of a call to a service very similar to a function, subroutine or method written in an unusual way compared to the norm and with the above mentioned call methods, useful in terms of interoperability in a typical complex modular architecture. Some of the advantages that can be achieved with the use of Web services are as follows:

- they allow interoperability between different software applications on different hardware platforms;

- they use "open" standards and protocols; the protocols and data format is, where possible, in textual format, which makes them easier to understand and use by developers;

- through the use of HTTP for message transport, Web services normally do not require changes to security rules used as filters on firewalls;

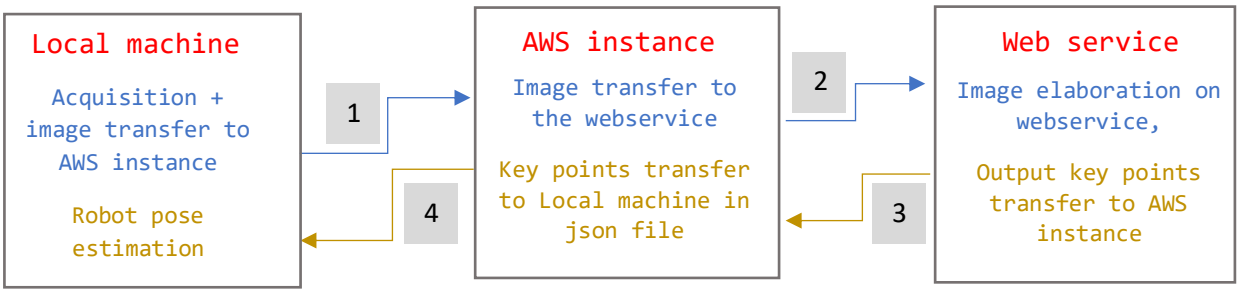
- can easily be used in combination with one another (regardless of who provides them and where they are made available) to form complex, "integrated" services;

- they allow the reuse of already developed infrastructures and applications and are (relatively) independent from any changes to them;

- they hide to the customer the architectural complexity of the framework, offering in an eventual interface (front end) the result of the execution of the service (as an example the values of determined fields of a table in an app).

It is desired to obtain a flow of information and processing that is adaptable and independent of the physical machine on which it is requested.

The following is a diagram of the structure of the end system:



1) Local acquisition, image is sent to AWS instance

2) AWS sends image to webservice

3) AWS receive output key points from webservice

4) AWS sends key points on local machine

In addition to the libraries used for local processing it was chosen to use the request library: Requests is an elegant and simple HTTP library for Python, built for human beings. Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3.

Considering the flow shown in the figure the code in this phase is implemented in 3 main parts to get the image processing and the robot response:

1. Local

1.a Image capture

```
### Acquisition from depth-camera ###
start_time = time.time()
pipe = rs.pipeline()

cfg = rs.config()
profile = pipe.start(cfg)
depth_sensor = profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()
print("Depth scale = ", depth_scale)

for x in range(5):
    pipe.wait_for_frames()

frameset = pipe.wait_for_frames()
color_frame = frameset.get_color_frame()

color = np.asanyarray(color_frame.get_data())
colorizer = rs.colorizer()

align = rs.align(rs.stream.color)
frameset = align.process(frameset)
aligned_depth_frame = frameset.get_depth_frame()
```

```
colorized_depth =
np.asanyarray(colorizer.colorize(aligned_depth_frame).get_data())
```

1.b Image saving

```
imgD = Image.fromarray(colorized_depth)
imgRGB = Image.fromarray(color)
imgD.save("/home/anna/PycharmProjects/Final_project/depth1.jpg")
imgRGB.save("/home/anna/PycharmProjects/Final_project/RGB1.jpg")
```

1.c Image Transfer to Remote Instance

```
### Uploading .jpg file on Remote EC2 instance ### !!! IP changes each time
instance starts

trans_frame = 'scp -i EC2kp.pem RGB1.jpg  ubuntu@ec2-100-27-12-143.compute-
1.amazonaws.com:/home/ubuntu/flask_project/static/images'
call(trans_frame.split())
```

1.d Get keypoints from remote instance and compute the joint angle configuration

```
### UPLOAD keypoints in local from keypoints file ###
myArray =
np.load(open('/home/anna/PycharmProjects/Final_project/keypoints.npy',
'rb'),allow_pickle=True)

nstr = myArray[0]
sstr = myArray[1]
estr = myArray[2]
wstr = myArray[3]

narr = (nstr[2:(len(nstr)-2)]).split()
sarr = (sstr[2:(len(sstr)-2)]).split()
earr = (estr[2:(len(estr)-2)]).split()
warr = (wstr[2:(len(wstr)-2)]).split()

n = np.array([np.array([float(narr[0]), float(narr[1]), float(narr[2])]),])
s = np.array([np.array([float(sarr[0]), float(sarr[1]), float(sarr[2])]),])
e = np.array([np.array([float(earr[0]), float(earr[1]), float(earr[2])]),])
w = np.array([np.array([float(warr[0]), float(warr[1]), float(warr[2])]),])

# punti chiave salvati

print("Neck coordinates in absolute rf: \n" + str(n))
print("\nRight Shoulder coordinates in absolute rf: \n" + str(s))
print("\nRight Elbow coordinates in absolute rf: \n" + str(e))
print("\nRight Wrist coordinates in absolute rf: \n" + str(w))


cfg = pipe.start() # Start pipeline and get the configuration it found
profile = cfg.get_stream(rs.stream.depth) # Fetch stream profile for depth
stream
intr = profile.as_video_stream_profile().get_intrinsics() # Downcast to
video_stream_profile and fetch intrinsics
print(intr.ppx)
```

```
print(intr.ppy)
print(intr.fx)
print(intr.fy)

depth_image = np.asanyarray(aligned_depth_frame.get_data())
depth = depth_image[np.int(intr.ppx),np.int(intr.ppy)].astype(float)
distance = depth * depth_scale
print("distance = ", distance)
pipe.stop()


Mi = np.array([[intr.fx, 0, intr.ppx], [0, intr.fy, intr.ppy], [0, 0, 1]])
Me = np.array([[1, 0, 0], [0, 0, 1], [0, -1, 0]])
M_p_cm = np.linalg.inv(np.dot(Mi,Me))

print(M_p_cm)

# coordinate omogenee di neck, shoulder, elbow e wrist (CAMERA rf)
nq = np.array([n[0,0],n[0,1],1])
sq = np.array([s[0,0],s[0,1],1])
eq = np.array([e[0,0],e[0,1],1])
wq = np.array([w[0,0],w[0,1],1])

nxy_cm = np.dot(M_p_cm, nq)
sxy_cm = np.dot(M_p_cm, sq)
exy_cm = np.dot(M_p_cm, eq)
wxy_cm = np.dot(M_p_cm, wq)

nxp = np.int(n[0,0])
nyp = np.int(n[0,1])
depth_nz = depth_image[nyp,nxp].astype(float)
nz_cm = depth_nz * depth_scale

sxp = np.int(s[0,0])
syp = np.int(s[0,1])
depth_sz = depth_image[syp,sxp].astype(float)
sz_cm = depth_sz * depth_scale

exp = np.int(e[0,0])
eyp = np.int(e[0,1])
depth_ez = depth_image[eyp,exp].astype(float)
ez_cm = depth_ez * depth_scale

wxp = np.int(w[0,0])
wyp = np.int(w[0,1])
depth_wz = depth_image[wyp,wxp].astype(float)
wz_cm = depth_wz * depth_scale

n_cmABS = np.array([nxy_cm[0],nxy_cm[2],nz_cm])
s_cmABS = np.array([sxy_cm[0],sxy_cm[2],sz_cm])
e_cmABS = np.array([exy_cm[0],exy_cm[2],ez_cm])
w_cmABS = np.array([wxy_cm[0],wxy_cm[2],wz_cm])

TT = np.array([[1,0,0,0], [0,0,1,0], [0,-1,0,0], [0,0,0,1]])
# neck as reference
n_cm = np.dot(TT,np.array([0,0,0,1]))
s_cm = np.dot(TT,np.array([sxy_cm[0]-nxy_cm[0],sxy_cm[2]-nxy_cm[2],sz_cm-
nz_cm,1]))
e_cm = np.dot(TT,np.array([exy_cm[0]-nxy_cm[0],exy_cm[2]-nxy_cm[2],ez_cm-
nz_cm,1]))
w_cm = np.dot(TT,np.array([wxy_cm[0]-nxy_cm[0],wxy_cm[2]-nxy_cm[2],wz_cm-
```

```
nz_cm,1]))


print(n_cm)
print(s_cm)
print(e_cm)
print(w_cm)

#model J1
ns_norm = np.linalg.norm(n_cm-s_cm)
ns = [0,0,ns_norm,1]
print("starting pose: \n", ns)
m1 = tinyik.Actuator(['z', ns[0:3]])
target1 = np.array([s_cm[0],s_cm[1],0])
print("\n target point1 xz: \n", target1)
m1.ee = target1
a1 = m1.angles
print("\n J1 =", np.rad2deg(a1))

#rotation matrix J2
R1 = np.array([[cos(a1), -sin(a1), 0, 0], [sin(a1), cos(a1), 0, 0], [0, 0,
1, 0], [0, 0, 0, 1]])

#model J1
ns_rot = np.dot(R1,ns)
print("\n starting pose after J2: \n", ns_rot)
m2 = tinyik.Actuator(['y', ns_rot[0:3]])
target2 = np.array([s_cm[0],0,s_cm[2]])
print("\n target point2 xy: \n", target2)
m2.ee = target2
a2 = m2.angles
print("\n J2 =", np.rad2deg(a2))

R2 = np.array([[cos(a2), 0, sin(a2), 0], [0, 1, 0, 0], [-sin(a2), 0,
cos(a2), 0], [0, 0, 0, 1]])

# next joints
Rt0 = np.dot(R1,R2)
R12 = np.transpose(Rt0)
t12 = -np.dot(R12,s_cm)
T12 = np.array([[R12[0,0], R12[0,1], R12[0,2], t12[0]],[R12[1,0], R12[1,1],
R12[1,2], t12[1]], [R12[2,0], R12[2,1], R12[2,2], t12[2]], [0, 0, 0, 1]])
print("origin in shoulder: \n", np.round(np.dot(T12,s_cm)))
print("elbow coordinates in new rf: \n", np.dot(T12,e_cm))

#model J3
es_norm = np.linalg.norm(e_cm-s_cm)
es = [0,0,es_norm,1]
e_srf = np.dot(T12,e_cm)
m3 = tinyik.Actuator(['y', es[0:3]])
target3 = np.array([e_srf[0], 0, e_srf[2]])
print("target point 3 xz: \n", target3)
m3.ee = target3
a3 = m3.angles
print("\n J3 = ", np.rad2deg(a3))

#rotation J3
R3 = np.array([[cos(a3), 0, sin(a3), 0], [0, 1, 0, 0], [-sin(a3), 0,
cos(a3), 0], [0, 0, 0, 1]])

#model j4
```

```
es_rot = np.dot(R3,[0,0,e_srf[2],1])
print("\n starting pose after J3: \n", es_rot)
m4 = tinyik.Actuator(['z', es_rot[0:3]])
target4 = np.dot(R3,np.array([e_srf[0],e_srf[1],0,1]))
print("\n target point 4 xy: \n", target4)
m4.ee = target4[0:3]
a4 = m4.angles
print("\n J4 = ", np.rad2deg(a4))

R4 = np.array([[cos(a4), -sin(a4), 0, 0], [sin(a4), cos(a4), 0, 0], [0, 0,
1, 0], [0, 0, 0, 1]])
Rt1 = np.dot(R3,R4)
Rt2 = np.dot(Rt0,Rt1)
R12_34 = np.transpose(Rt2)
t12_34 = -np.dot(R12_34, e_cm)
T12_34 = np.array([[R12_34[0,0], R12_34[0,1], R12_34[0,2],
t12_34[0]],[R12_34[1,0], R12_34[1,1], R12_34[1,2], t12_34[1]],
[R12_34[2,0], R12_34[2,1], R12_34[2,2], t12_34[2]], [0, 0, 0, 1]])
print("\n origin in elbow: \n", np.round(np.dot(T12_34,e_cm)))
print("\n wrist coordinates in new rf: \n", np.dot(T12_34,w_cm))

#model J5
ew_norm = np.linalg.norm(e_cm-w_cm)
ew = [0,0,ew_norm,1]
w_erf = np.dot(T12_34,w_cm)
m5 = tinyik.Actuator(['y', ew[0:3]])
target5 = np.array([w_erf[0], 0, w_erf[2]])
print("\n target point 5 xz: \n", target5)
m5.ee = target5
a5 = m5.angles
print("\n J5 = ", np.rad2deg(a5))

#rotation J5
R5 = np.array([[cos(a5), 0, sin(a5), 0], [0, 1, 0, 0], [-sin(a5), 0,
cos(a5), 0], [0, 0, 0, 1]])

#model j6
ew_rot = np.dot(R5,[0,0,w_erf[2],1])
print("\n starting pose after J5: \n", ew_rot)
m6 = tinyik.Actuator(['z', ew_rot[0:3]])
target6 = np.dot(R5, np.array([w_erf[0],w_erf[1],0,1]))
print("\n target point 6 xy: \n", target6)
m6.ee = target6[0:3]
a6 = m6.angles
print("\n J6 = ", np.rad2deg(a6))

print('\nj1 su z: ', np.rad2deg(a1))
print('j2 su y: ', np.rad2deg(a2)-np.rad2deg(a2))
print('j3 su y: ', np.rad2deg(a3)+np.rad2deg(a2))
print('j4 su z: ', np.rad2deg(a4))
print('j5 su y: ', np.rad2deg(a5))
print('j6 su z: ', np.rad2deg(a6))


#myedo.movejoint(j1=,j2=,j3=,j4=,j5=,j6=)
```

2. AWS instance

2.a Image transfer to webservice

```
### Openpose called on webservice by remote EC2 instance ###  !!! IP
changes each time instance starts

rem_script = 'ssh -i EC2kp.pem  ubuntu@ec2-100-27-12-143.compute-
1.amazonaws.com python3 < app.py'
call(rem_script.split())
```

2.b Get webservice output

```
n,s,e,w=find_keypoints()

myList=np.array([n,s,e,w])
np.array(myList).dump(open('/home/ubuntu/keypoints.npy', 'wb'))
```

2.c Local file transfer

```
### Loading .jpg file on local ### !!! IP changes each time instance starts

trans_kp = 'scp -i EC2kp.pem  ubuntu@ec2-100-27-12-143.compute-
1.amazonaws.com:/home/ubuntu/keypoints.npy
/home/anna/PycharmProjects/Final_project'
call(trans_kp.split())
```

3. Web Service

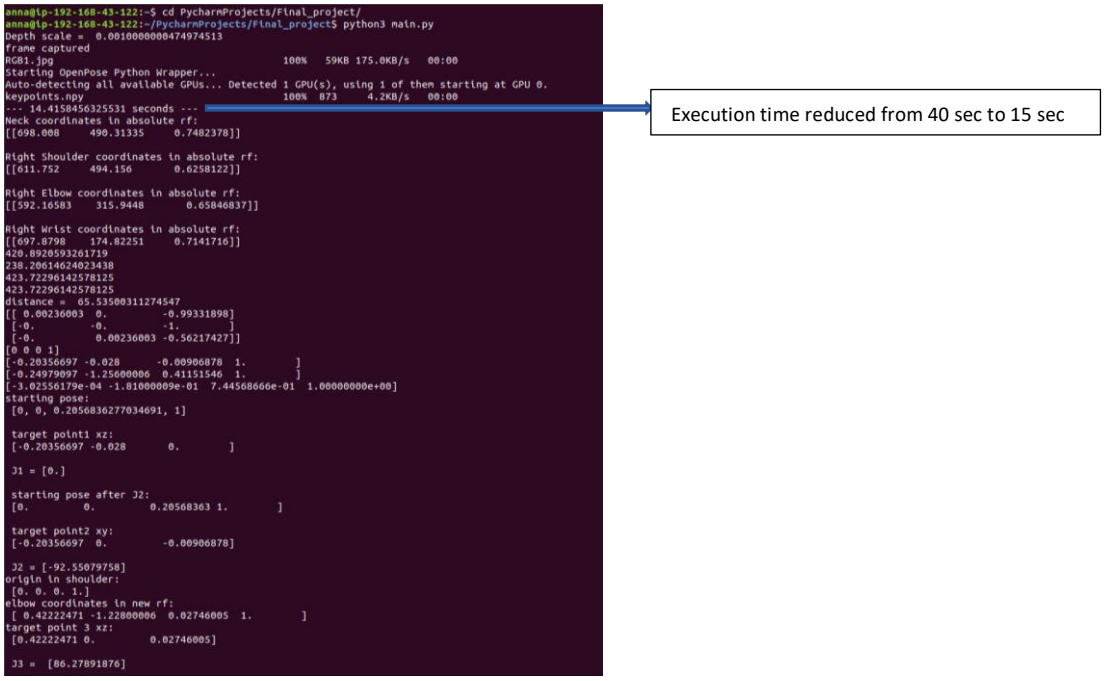
3.a Points detection

```
# app
app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

@app.route('/')
def find_keypoints(filename='RGB1.jpg'):
    # open and process image
    target = os.path.join(APP_ROOT,
'/home/ubuntu/flask_project/static/images')
    destination = "/".join([target, filename])

    parser = argparse.ArgumentParser()
    parser.add_argument("--image_path", default=destination, help="Process
an image. Read all standard formats (jpg, png, bpm, etc.).")
    parser.add_argument("--no_display", default=False, help="Enable to
disable the visual display.")
    args = parser.parse_known_args()

    # Custom Params (refer to include/openpose/flags.hpp for more
parameters)
    params = dict()
    params["model_folder"] = "/home/ubuntu/openpose/models/"
```

```
    opWrapper = op.WrapperPython()
    opWrapper.configure(params)
    opWrapper.start()

    datum = op.Datum()
    imageToProcess = cv2.imread(args[0].image_path)
    datum.cvInputData = imageToProcess
    x = op.VectorDatum([datum])

    result = opWrapper.emplaceAndPop(x)

    n = str(datum.poseKeypoints[:, 1, :])
    s = str(datum.poseKeypoints[:, 2, :])
    e = str(datum.poseKeypoints[:, 3, :])
    w = str(datum.poseKeypoints[:, 4, :])

    return n,s,e,w

n,s,e,w=find_keypoints()

myList=np.array([n,s,e,w])
np.array(myList).dump(open('/home/ubuntu/keypoints.npy', 'wb'))
```

10.3 Results after AWS and web service integration

From the test performed on the machine the execution time is reduced by about 60%.



The result obtained does not reflect the expected one, with the right hardware structure we expected a real time frame processing. The purpose of the test was to obtain an improvement in processing for the purpose of processing video acquisitions. With the obtained result it is still

impossible to use a sequence of frames for the motion demonstration. We stop at the demonstration acquired for single images, usable in composition to program a complete action.

11. Conclusions

The objective of this thesis project was to obtain a robotic support system, which could collaborate with the user during the performance of basic actions such as the mutual exchange of objects. Specifically, the strength of the project is in terms of programming of the same, in fact, the implementation of the system in its entirety aims to relieve the user from tasks of computer / robotic type.

The programming by imitation has as its ultimate goal to give the user to use and modify the tasks for which a robot is programmed without the need to know in depth the programming techniques at the base of the system.

In addition to specific technical features that would have given the system adaptability to any user, the project aimed to remain in a low cost development range, considering the chosen manipulator around which the global architecture of the system revolves.

During the drafting of the project and the development of the control code, the proposal was to remain below the price ranges typical of automation systems used to perform pick and place tasks (industrial manipulators).

The choice of the system components was made following this principle and during the development of the system some problems emerged caused by the compromise that was chosen to follow.

The main limitations identified during the development of the system can be distinguished according to the system component they refer to:

- Manipulator

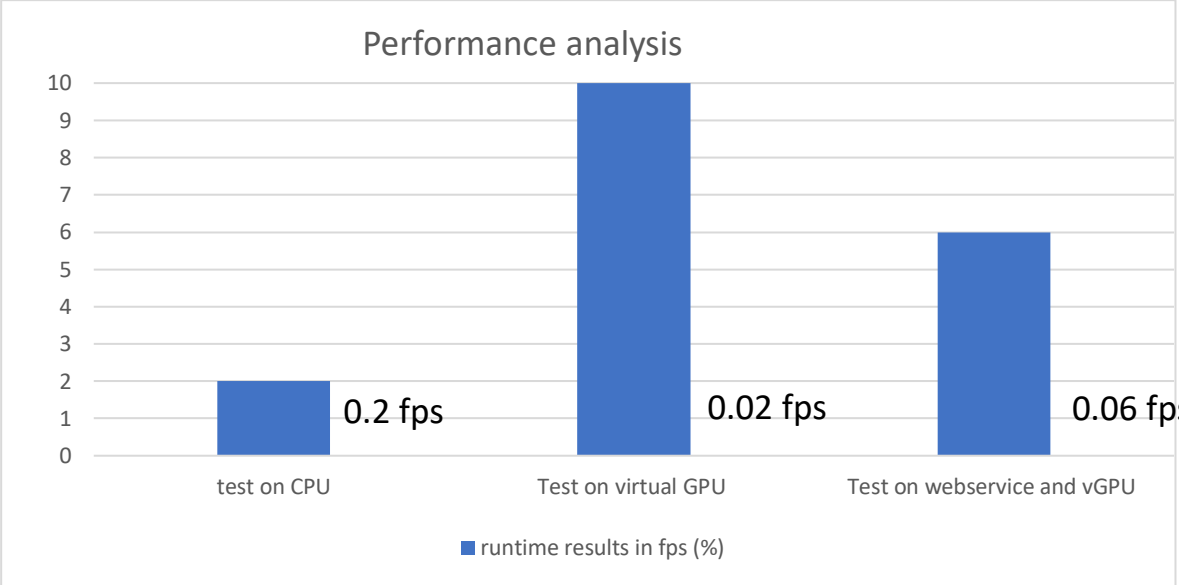
The chosen manipulator is a manipulator used for educational and recreational purposes. The accuracy of the motion is not on par with the accuracy that characterizes industrial manipulators. In the manipulator design, the motors in the joints carry a low load. The joints at the base have a lower stress margin than the upper ones; in fact, the manipulator has a control-braking system that stops the movement in progress if the stress exceeds the threshold supported by the specific joint. In order to obtain movements similar to those of humans, the joints at the bottom should be able to support inclinations greater than those contemplated. This limitation forces the adoption of a (fixed) angle imposition at joint 2 in order to avoid the activation of this control system during imitation. Based on this imposition, the imitation appears less truthful than the movement shown by the user.

- PC graphics processor

Deep learning based systems are based on towed neural networks that need the right hardware support to produce usable and applicable results in automated processes. In the case of the implemented system, the processor of the PC on which the development has been done does not have a dedicated graphics card: this involves the impossibility to obtain results in real time and therefore usable for applications on the market. To stay in the theme of low cost development, AWS systems have been fundamental to obtain much better results, even if still characterized by a finite latency time.

- Web services

The failure to obtain real-time processing may have been caused not only by the use of virtual hardware and not physical, but also by the use of the web service, which, although it gives versatility to the use of the system, has lower performance than those found using alternative approaches to distributed computing such as Java RMI, CORBA, or DCOM.



Although the system obtained is not usable in real time, the development of this thesis has brought to light several positive evaluations, in terms of compatibility between systems of different nature.

The primary meaning of collaborative robotics puts next to the traditional robotics the interaction with the user: this implies the use of artificial intelligence that gives sensory perception to the system and allows to have control over the safety of the user and the tasks carried out in collaboration. In order to obtain polyvalent systems of this type, it is necessary to base the implementation choices on the versatility and adaptability of the instrumentation and programming languages used. Most of the problems that emerged during development were easily overcome when compatibility between systems made it possible. It becomes fundamental for purposes like the one treated, to carry out a qualitative analysis in terms of compatibility, upstream of the implementation of the global system.

Bibliography

1. Robot Programming by Demonstration
   – Authors: Aude Billard, Sylvain Calinon, Rüdiger Dillmann, Stefan Schaal
2. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields
   – Authors: Zhe Cao, Student Member, IEEE, Gines Hidalgo, Student Member, IEEE,Tomas Simon, Shih-En Wei and Yaser Sheikh
3. A methodology to develop collaborative robotic cyber physical systems for production environment – Authors: Azfar Khalid, Pierre Kirisci, Zied Ghrairi, Klaus-Dieter Thoben, Ju¨rgen Pannek
4. A computationally efficient safety assessment for collaborative robotics applications
   – Authors: Matteo Parigi Polverini, Andrea Maria Zanchettin, Paolo Rocco
5. Dynamic task classification and assignment for the management of human-robot collaborative teams in workcells
   – Authors: Giulia Bruno · Dario Antonelli
6. Industrial Robot: An International Journal
   – Autor: Richard Bloss
7. A Brief History of Machine Learning
   – Author: Keith D. Foote
8. A Brief History of Deep Learning
   – Author: Keith D. Foote
9. Applied Deep Learning - Part 4: Convolutional Neural Networks
   – Author: Arden Dertat
10. Human Tracking Using Convolutional Neural Network
    – Authors: Jialue Fan, Wei Xu, Yihong Gong
11. Human Object Identification for Human-Robot Interaction by using Fast R-CNN
    – Authors: Shih-Chung Hsu, Yu-Wen Wang
12. Artificial Intelligence and Robotics
    – Authors: Javier Andreu Perez, Fani Deligianni, Dnaiele Ravi and Guang-Zhong Yang
13. Robot Learning from Demonstration in Robotic Assembly
    – Authors: Zuyuan Zhu, Huosheng Hu
14. Intel RealSense Stereoscopic Depth Cameras
    – Authors: Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, Achintya Bohwmik
15. Real-Time Computer Stereo Vision for Automotive Applications

– Authors: Dr. Naim Dahnoun, Prof. John G. Rarity
16. Texture Synthesis Repair of RealSense D435i Depth Images with Object-Oriented RGB Image Segmentation – Authors: Longyu Zhang, Hao Xia, and Yanyou Qiao
17. https://www.intelrealsense.com/depth-camera-d435i/
18. https://towardsdatascience.com/choosing-the-right-gpu-for-deep-learning-on-aws-d69c157d8c86
19. Webb G.I. (2011) Overfitting. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA.