Supporting Impaired People with a Following Robotic Assistant by means of End-to-End Visual Target Navigation and Reinforcement Learning Approaches

Nguyen Ngoc Dat[1], Valerio Ponzi[1], Samuele Russo[2] and Francesco Vincelli[1]

[1]*Department of Computer, Control and Management Engineering, Sapienza Univeriisty of Rome, 00185 Rome, Italy*
[2]*Department of Psychology, Sapienza University of Rome, via dei Marsi 78 Roma 00185, Italy*

Abstract
We present an improvement in visual object tracking and navigation for mobile robot implementing the advantage actor-critic (A2C) reinforcement learning architecture on top of the Gym-Gazebo framework. This work provides an easier way to integrate reinforcement learning algorithms for navigation and object tracking tasks in robotics field. We train the convolutional-recurrent model employed for the policy estimation in an end-to-end manner. The robot is able to follow a simulated human walking in an indoor environment by using the sequence of images provided by the robot camera. The input of the algorithm is acquired and processed directly in ROS-Gazebo environment. The policy learned by the robot agent proved to generalize well also in an environment with different size and shape with respect to the training one. Moreover, the policy allows the robot to avoid obstacles while following the tracking target. Thanks to these improvements, we can straightforwardly apply the tracking system in a real world robot for a person following task in indoor environments.

Keywords
Visual Object Tracking, Human Tracking, Person Following, Human Robot Interaction, ROS, Gazebo, Gym-Gazebo, Visual Navigation, Reinforcement Learning, Advantage Actor Critic

1. Introduction

In some categories of subjects defined as "fragile" the presence of an assistant who can guide and help them can be a valid help. In some cases, this assistant is not only useful but also necessary, especially with subjects who show spatial orientation problems. The idea of this study concerns the accompaniment of the person to a robot that accompanies and helps him when it is necessary to regain the visuospatial orientation. For example, when the subject walks down the street, the robot recognizes if that is the right path and, if so, communicates to the person that he has taken the wrong path and suggests the appropriate path. This functionality is very useful and sometimes indispensable when the patient has cognitive problems that can have implications on executive functions and on the abilities of visuospatial orientation [1, 2, 3]. Where self-monitoring and visuospatial planning skills are lacking, the use of this robot can help the patient on the one hand to maintain greater autonomy and independence, and on the other hand to reinforce, strengthen and rehabilitate skills. visuospatial. In fact, the use of the robot would not be limited only to a replacement of the action that the person has failed, but it would also be able to help the person reflect on the mistake made, asking the person questions that help him to reflect and reason. . Furthermore, the activation of reasoning can occur even before the person is about to commit a visual-spatial error. For example, the robot can prompt the person to activate a reasoning based on the planning of the path that is shorter or better reachable. Furthermore, the robot can also perform the function of "companion" with which the person talks, and then chooses and decides independently with the help of the robot's questions, which is the place where he prefers to go. In fact, in patients with mild cognitive impairment, it is easy for the subject to act on impulse, without adequate planning of the path and without having reflected on the objectives for which one chooses to follow that path. From a psychological and neuropsychological point of view, the use of robots plays a fundamental role in supporting, on the one hand, the self-determination and autonomy of the person and on the other, slowing down their decline and activating neuropsychological enhancement processes mediated by the robot. Object detection is the process of detecting the object in frames of a video sequence, while the object tracking is the process of finding the direction of an object while moving around a scene [4] The main steps of the tracking process are: (1) Detection of moving objects, (2) Tracking of the related object from the current frame to the next frame, (3) Analysis of tracking objects to recognize their behaviour. Visual tracking plays an important role in fusing many computer vision applications which include image and video processing, pattern recog-

ICYRIME 2021 @ International Conference of Yearly Reports on Informatics Mathematics and Engineering, online, July 9, 2021
ponzi@diag.uniroma1.it (V. Ponzi); samuele.russo@uniroma1.it (S. Russo)
0000-0002-1846-9996 (S. Russo)

nition, information retrieval, automation and control. The tracking procedure finds itself in many applications like mobile robotics, solar forecasting, particle tracking in microscopy images, biological applications, surveillance, to cite the most common ones [5, 6]. Much of the existing work related to object tracking is on passive tracker, where it is assumed that the object of interest is always in the image scene, and there is no need to handle camera control during tracking. This approach is not suitable for some use-cases, e.g., the tracking performed by a mobile robot with a camera mounted or by a drone. For such applications, one should seek a solution to approach active tracking, which unifies the two sub-tasks, i.e., the object tracking and the camera control. In the passive tracker approach, it is difficult to jointly tune the pipeline with the two separate sub-tasks. The tracking task may also involve many human efforts for bounding box labeling. Moreover, the implementation of camera control is non-trivial and can incur many expensive trial-and-errors system tuning in the real-world, as shown in [7], [8]. Active object tracking additionally considers camera control compared with traditional object tracking. There exists not much research focus in this approach for visual object tracking so far.

2. Related Works

Despite the success of traditional trackers based on low-level, hand-crafted features, models based on deep convolutional neural network (CNN) have dominated recent visual tracking research. The success of these models largely depends on the capability of CNN to learn a good feature representation for the tracking target. Unfortunately, for a busy scene with occluding objects, this approach can fail to find long-term temporal correlations expressing target motion along different frames. In this work we explore and investigate a more general strategy to develop a novel visual tracking approach based on reinforcement learning and convolutional recurrent networks. The major intuition behind this method is that, during the active tracking process, the historical visual semantics and tracking proposals encode pertinent information for future predictions. Such features require continuous and accurate predictions in both spatial and temporal domain over a long period of time, thus demanding for a novel network architecture design as well as proper training algorithms. We formulate the visual tracking problem as a sequential decision-making process and explored a novel framework, referred to as Deep RL Tracker (DRLT). The latter processes video frames as a whole and directly outputs actions to make the camera able to follow the target in each frame. Our model integrates convolutional network with recurrent network (Figure 2), and builds up a spatial-temporal representation of the input frames. It fuses past recurrent states with current visual features to make predictions of the target object's movements along the input sequence of frames over time. We employed an end-to-end algorithm that allows the model to be trained to maximize tracking performance in the long run. This procedure uses back-propagation to train the neural network components and off-policy actor-critic reinforcement learning algorithm [9] to train the policy network. Recent research in visual object tracking relies on game engines as simulation environment to perform training of the neural network models to be then applied on physical robotic platform and real-world environments. We notice that game engines are not suitable for mobile robot applications, such as the person following the task considered in this work. Game engines only allow to control the camera position and orientation, without caring about how to control the robot motion and navigation in response to tracking outputs, since the game engine does not have any robot hardware APIs. For this reason, our approach relies on a simulation environment based on ROS/Gazebo framework for the training process, providing suitable APIs to deal with camera control by navigation of the robotic platform carrying the camera sensor. Our main target is teaching the mobile manipulator TIAGo, from PAL Robotics, to follow a human target, while walking in an indoor environment, for assistance and health-care task. TIAGo robot and the human tracking target are modeled in Gazebo 3D simulator. A sequence of images, acquired by robot camera sensor, is passed as input to the *observation encoder*; then the *sequence encoder* collects and encodes the temporal correlation of extracted feature representation; after these steps, the advantage actor-critic (A2C) RL off-policy algorithm is used to optimize the actor and critic networks through policy gradient and value loss and the output of reinforcement learning algorithm is then used to sample the new action which the robot has to perform to follow the human trajectory.

Target-driven visual navigation is a relatively new task in the field of robotics research. Only recently, end-to-end systems have been specifically developed to address this problem. A possible naive approach could be to use a classic map-based navigation algorithm along with an image or object recognition model.
To overcome these limits, map-less methods, which try to solve the problem of navigation and target approaching jointly, have been proposed [10, 11]. These systems, like ours, do not build a geometric map of the area, instead, they implicitly acquire the minimum knowledge of the environment necessary for navigation. This is done by direct mapping visual inputs to motion, i.e. pixels to actions. The DRL framework proves very promising for this purpose. In deep reinforcement learning and agent-based models, a reward function is defined based on the robot's perceived state and performed actions. The robot

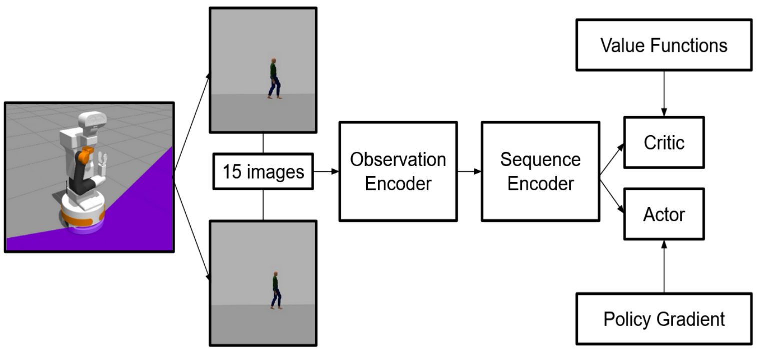


Figure 1: Overview of the network architecture. The encoder block contains 4 convolutional layers. The sequence encoder is a LSTM layer, which extract feature over time. The actor-critic networks are fully connected layers

learns sequential decision making to accumulate more rewards while in operation. The overall problem is typically formulated as a Markov decision process (MDP) and the optimal action-state rules are learned using dynamic programming techniques. These methods are attractive because they do not require supervision and they imitate the natural human learning experience. However, they require complex and lengthy learning processes. DRL for Robotic Applications and Visual Navigation RL is becoming popular in recent times. In [12] it is proposed a solution to the navigation problem of nonholonomic mobile robots with continuous control based on deep-RL. Moreover, training the robot for the motion task in a virtual environment allows to speed up the learning and generalization process and also avoid the costs and risks of a trial and error learning approach in the real-world setup. Equipped with deep ConvNets, RL shows impressive successes on visual tracking tasks as shown also in [13]. However, they are distinct from this work, as they do not formulate the tracking procedure in an end-to-end manner and do not consider camera controls. A further step towards generalization is taken by [14], which introduces a framework that integrates a deep neural network based object recognition module. With this module, the agent can identify the target object regardless of where the photo of that object is taken. However, it is still trained or fine-tuned in the same environments where it is tested. Therefore, it is still not able to generalize to unseen scenarios.

3. System set-up and simulation environment

The first step in the development of this work consisted in the setup of a simulation environment in which the TIAGo robot model and a walking human model (actor) can be reproduced. As described in the previous chapter, the task of the project is the development and the improvement of a visual object tracking system allowing to actively track a human walking in an indoor environment in an end-to-end manner by means of an actor-critic RL algorithm. To avoid the potential issues about the different operative systems running on our machines, the continuous update of dependencies and deprecated packages, which can prove troublesome for developing while using a *Robot Operating System (ROS)* code-base, a Docker container has been built to develop, run, manage and sync the modules building up the whole project (refer to paragraph 3.1) The Gazebo was chosen as a simulation environment since it can plugin directly to the ROS framework. Robot simulation is an essential tool in every robotics toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. For the integration of RL algorithm in ROS/Gazebo framework we relied on *Gym-Gazebo*, a toolkit which extends the *OpenAI Gym* for robotics, providing different learning techniques and algorithms to be compared using the same virtual conditions.

3.1. Docker

Docker is a software platform that allows one to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. We managed to build a complex Docker container, requiring to sync many packages and tools that are dependent to each other. For instance, TIAGo robot ROS package (`tiago_public_ws`) require *ROS Melodic* version, ROS Melodic requires *Python v2.7*; about the RL module, *Gym-Gazebo* requires PyTorch machine learning framework

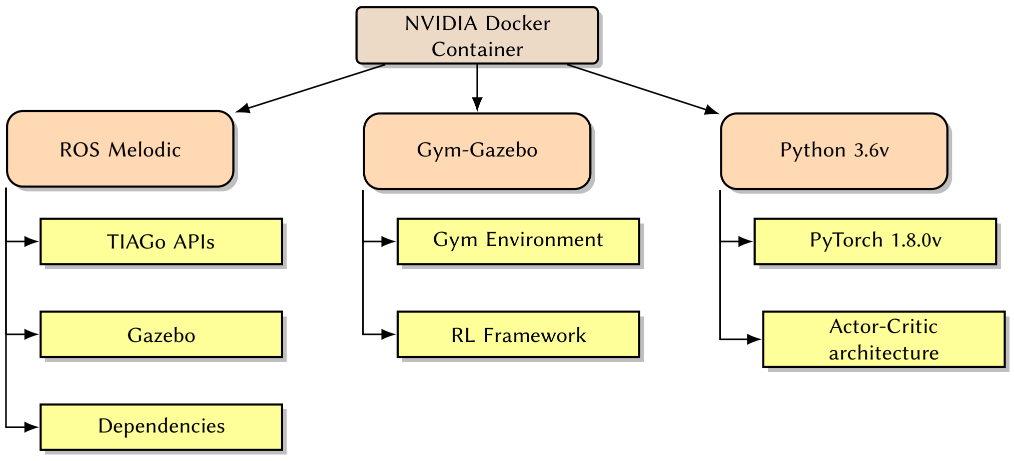


Figure 2: Scheme of the project framework in docker container

and PyTorch requires *Python v3.6*. Thus we need to setup both python2 and python3 for adapting with the system. Having an NVIDIA GPU available, we leverage the *NVIDIA Container Toolkit* that allows users to build and run GPU-accelerated Docker containers.

GPU-enabled applications need access to both kernel-level device drivers and user-level CUDA libraries, and different applications may require different CUDA versions. One way to solve this problem is to install the GPU drivers inside the container and map the physical NVIDIA GPU device on the underlying Docker host (e.g., /dev/nvidia0) to the container. The problem with this approach is that the version of the driver and libraries inside the container needs to precisely match. Otherwise, the application will fail. In such a case, users still have to worry about what drivers and libraries are installed on each host computer to ensure compatibility with containerized applications.

NVIDIA Docker, instead, provides driver-agnostic CUDA images. This Docker plug-in enables GPU applications running in containers to share graphic acceleration devices on the Docker host without worrying about version mismatches between libraries and device drivers. In particular, we employed the *rocker toolkit* [15] that, building upon the nvidia-docker2 package, provides an easy way to run the docker container with graphic user interface and GPU acceleration. The structure of the docker container is shown in detail in the diagram of Figure 2

3.2. ROS - Gazebo

ROS is a collection of libraries, drivers, and tools for the effective development and building of a robot systems. It has a Linux-like command tool, an inter-process communication system, and numerous application-related packages. The main features of the ROS infrastructure are: **Nodes**. They are the executable processes that participate in the communication. They can be programmed in C ++ or Python. For this project, Python has been chosen for its speed and simplicity and the built-in integration with the PyTorch deep learning library. **Topics**. The inter-process communication has a Publish/Subscribe model and the communication data are called Topics. They are the communication channels, defined by a specific name and a single type of message that can be posted to them. Nodes can subscribe to topics to receive the information published in them or publish information for other nodes to receive it. **Callbacks**. They are the interruption service routines that are generated when a node subscribed to a topic detects that something has been published on that channel. In this routine, the data processing is done, such as saving the position of the robot in the callback generated by the topic where an odometry sensor publishes the readings. **Launch**. A class of files that run and manage multiple nodes for the robot and its sensors, the simulation environment and the data visualization software, such as *Rviz* or Gazebo simulator. They are encoded in XML format.

URDF (Undefined Robot Description Format). They are definition files of a robot. The links of the robot's kinematic structure are connected to each other by joints. They also define the dynamic, kinematic, visual and collision properties of the robot. As launch files, they are programmed in XML too.

Gazebo is a simulator specifically designed for robotics. Its convenient design makes it possible to quickly test algorithms, robots and AI applications. In Gazebo all the elements present in reality are simulated, the sensors and actuators act according to the environment. In this work, Gazebo has been used for the design of the virtual indoor environment, where all the experiments with the TIAGo robot are performed.

3.3. TIAGo Robot

To better fit our task of the person following via visual object tracking algorithm, we consider some robot platforms, such as *TIAGo* and *Turltebot*. Differently from TIAGo, Turltebot provides with APIs, packages and open source ROS plugins. However, we chose TIAGo robot platform that, although it does not provide as many open source APIs as Turltebot, it is a more advanced and modern robot platform and its kinematic structure can better adapt to our target. Since the aim of our work is to make a mobile robot to be able to learn in an end-to-end manner how to actively detect and track a person walking in an indoor space, the height of TIAGo is more suitable because the camera mounted on the head of the robot can catch the full human body of the person to follow, thus providing better input data to the reinforcement learning algorithm. Moreover, the robot is a service robot specifically designed to work in indoor environments, so our application can be easily deployed on the real platform and in the real world.
Following the main components of the TIAGo robot are described (see Figure 3).

- **The mobile base** uses a differential drive system and has a maximum speed of 1 m/s. It is designed for indoor operation. At the base is the laser sensor that has a variable sensing distance depending on the model (iron, steel or titanium), between 5.6 meters and 25 meters. To detect what lies behind the robot, there are 3 sonars of 1 meter of detection.

- **The body** is the central part of the TIAGo robot and is made up of the arm and torso. The torso has a prismatic articulation that allows increasing the height of the robot by 35cm. The arm has 7 degrees of freedom, a length of 87 cm in its maximum extension, and a load capacity of 3kg.

- **The head** comprises the neck, having 2 DoF that allow TIAGo to look in any direction. In the place of the eyes, there is an RGB-D camera that provides color and depth images, being able to recreate the environments using point clouds. The technology is the same as that used by the popular *Kinect* cameras. In particular, the robot is equipped with a built-in Asus Xtion camera.

PAL Robotics offers different ROS packages and libraries compatibility allowing TIAGo robot to perform complex perception, navigation, manipulation and human-robot interaction tasks. The platform is also equipped with a *Jetson TX2 Kit* that guarantees power-efficient computing resources well fitting to deep learning applications.

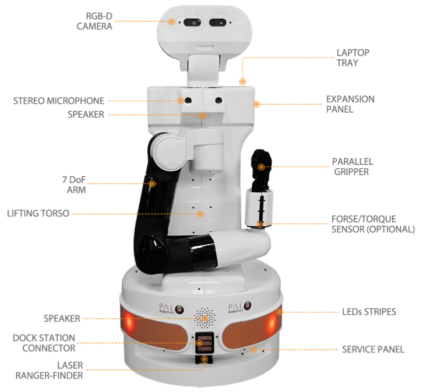


Figure 3: TIAGo robot with structural components highlighted)

3.4. Animated human model: actor

Gazebo simulator allows defining animated model (called 'actor' in Gazebo) which is useful if one wants to have entities following predefined paths in simulation without being affected by the physics engine. They have a 3D visualization that can be seen by RGB cameras, and 3D meshes which can be detected by GPU based depth sensors, so being suitable for computer vision applications. A closed-loop trajectory is defined for each of the considered train test cases. Additional plugin files are used to control animations based on feedback from the environment. Actors extend common models, adding animation capabilities. There are two types of animations that can be used separately or combined together:

- **Skeleton animation**, which is relative motion between links in one model;

- **Motion along a trajectory**, which carries all of the actor's links around the world, as one group;

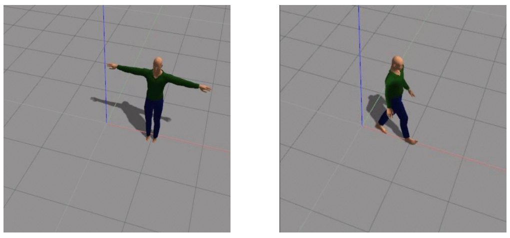
Both types of motions can be combined to achieve a skeleton animation that moves in the world. Gazebo supports two different skeleton animation file formats: COLLADA (`.dae`) and Biovision Hierarchy (`.bvh`). The actor model defined in the project loads a COLLADA file described within the `<skin>` tag. Sometimes, it is useful to combine different skins with different animations. Gazebo allows one to take the skin from one file, and the animation from another file, as long as they have compatible skeletons. **Scripted trajectories** represent the high-level animation type of actors, which consists of specifying a series of poses to be reached at specific times. Gazebo takes care of interpolating the motion between them so the movement is fluid. The trajectory is defined inside the `.world` file containing all the models which are visualized in simulation. Inside the `<script>` tag one can define some of the parameters for the desired trajectory, such as the *start time* and the if or not the motion has to be *repeated after it ends*. For our actor model's trajectory we set the trajectory to loop forever and start playing as soon as the world is loaded. In particular, the following parameters are available:

- **loop**: it is set to *true* for the script to be repeated in a loop. For a fluid, continuous motion is worth setting the last waypoint to match the first one;
- **delay_start**: this is the time in seconds to wait before starting the script. If running in a loop, this time will be waited before starting each cycle;
- **auto_start**: this attribute is set to *true* if the animation should start as soon as the simulation starts playing. It is useful to set this to *false* if the animation should only start playing only when triggered, for instance, by a plugin.

Finally, the actual trajectory is built by defining a sequence of *waypoints*: The parameter **waypoint** is described inside the `<trajectory>` tag and represents the intermediate targets the model has to reach along the trajectory path. Each waypoint consists of a *time* and a *pose*;

- **time** the time in seconds, counted from the beginning of the script, when the pose should be reached;
- **pose**: the pose which should be reached.

Once trajectories are created and static animations are loaded, the final step is to combine them in full synchronized animation and trajectory. Our skeleton animation contains a transnational component in the $x - axis$, as we could notice by running the animation without any trajectory. But that animation is not yet synchronized with our trajectory until we enable that by setting `<interpolate_x>` to *true* inside `<animation>` tag. Figure 4 shows how an actor is modeled and animated in Gazebo.



(a) Actor's skeleton (b) Actor's Animation

Figure 4: Building steps for actor's animation & trajectory

4. Reinforcement learning for robotics

Recent trends in reinforcement learning [16], show various applications in robotic fields, such as planning, control, air-based, under-water, land-based, etc. Moreover, a different state of the art RL techniques ad algorithms has been employed in this field, including actor-critic, deep reinforcement learning, and multi-agent learning. For the visual object tracking task this project deals with, two Actor-Critic methods have been considered: A2C (Advantage Actor Critic) in [17] and A3C (Asynchronous Advantage Actor Critic) in [18]. We decided to deploy the latter since it best fitted the available computational resources and the environmental settings. Thus the environment setup can be outlined as follows: a virtual environment implemented in Gazebo is combined with a reinforcement learning algorithm developed in ROS framework, encapsulating the agent description and reward updating in Gym-Gazebo package. Finally, a re-implementation of A2C algorithm using PyTorch library is used to define the policy learning method and manage the RL environment.

4.1. Gym gazebo

Gym-gazebo is a powerful toolkit for reinforcement learning of robotics applications that relies on ROS and Gazebo. This package follows the same baseline structure characterizing OpenAI Gym, and builds a ROS/Gazebo environment on top of that. Indeed, in the past years, non-profit AI research companies, such as OpenAI, have created a generic set of algorithm and environment interfaces. In OpenAI's Gym, agent-state combinations encapsulate information environments, which will be able to make use of all the available algorithms and tools. This abstraction allows easier implementation and tune of the RL algorithms, but most importantly, it creates the possibility of using any kind of virtual agent. This includes robotics, which Gym is already supporting with several environments on their roster. Gym-Gazebo toolkit aims to integrate the Gym APIs with robotic hardware, validating reinforcement learning algorithms in real environments. Real-world operation is achieved by combining Gazebo simulator with ROS, so providing a set of libraries and tools that help software developers to create robotic applications. The architecture consists of three main software blocks: OpenAI Gym, ROS and Gazebo. Environments developed

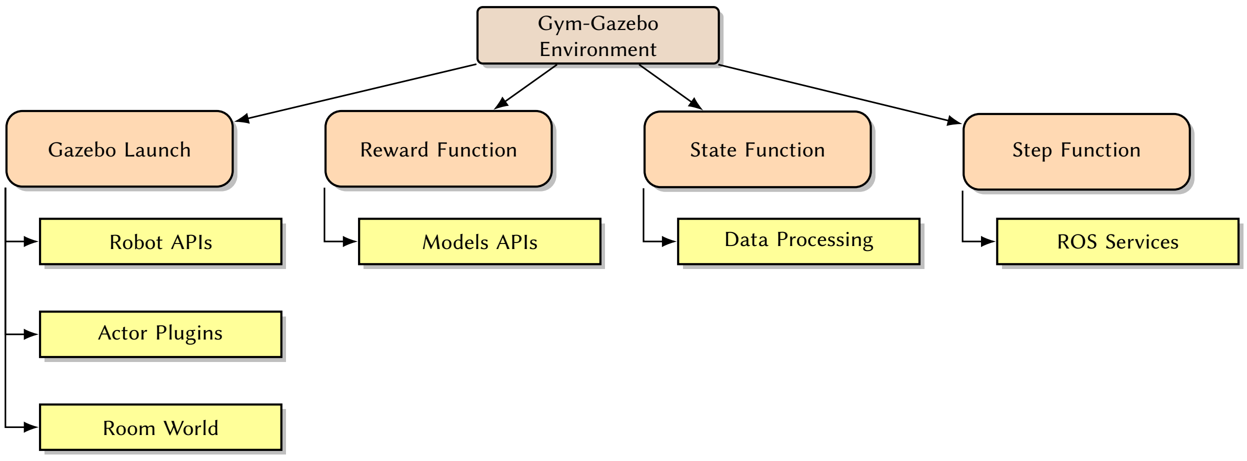


Figure 5: Gym-Gazebo Framework Architecture

in OpenAI Gym interact with the Robot Operating System, which is the connection between the Gym itself and Gazebo simulator. Gazebo provides a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.

We created our own environment for object tracking tasks and encapsulated it into Gym-gazebo package. There are two main observations to do for this step: first, we needed to correctly build the environment class following OpenAI Gym format; second, we had to fully integrate our own environment file inside the Gym-Gazebo toolkit, by inheriting the properties of python language. A common interface of OpenAI Gym environment provides three main methods to be implemented: `reset`, `step`, `render`. The function reset is used to re-initialize parameters for new episodes; the step function is called during training, and the render function is used to run Gazebo server. In Figure 5 it shows the architecture of the Gym-Gazebo toolkit and how it is extended for our implementation.

4.2. Reinforcement learning

Reinforcement Learning (RL) is an area of machine learning where a software agent learns by interacting with an environment, observing the results of these interactions with the aim of achieving the maximum possible cumulative reward. This imitates the trial-and-error method used by humans to learn, which consists of taking actions and receiving positive or negative feedback. In general, agent learning is based on a policy function that maps states to an action at each time step. The agent learns how to maximize the total rewards returned from the environment. **State space** - For the considered object tracking and human following task, the state consists in a sequence of RGB images, acquired from an action camera mounted on the robot head, and collected over time. The input sequence of images includes 15 camera frames sampled from a list collected in 0.5 seconds. Images are pre-processed, reshaped to size 128x128x3 and normalized before to be provided as input to the model. **Action space** - The agent action space is discrete and consists in three control commands: *go forward*, *rotate left* and *rotate right*, as described in more detail in the table Table 1, while the human walking speed along the trajectory is set to 0.55 m/s.

Table 1
Action space

Action	Forward	Rotate left	Rotate right
Linear velocity (m/s)	0.8	0.05	0.05
Angular velocity (rad/s)	0.0	0.7	0.7

Network model - The neural architecture defined for the implemented RL algorithm is shown in Figure 1. We experimented a more complex and deep neural model with respect to the network defined in [19], among the improvements proposed, the observation encoder contains more convolutional layer and also the size of features maps and kernels are changed. Moreover, the actor-critic framework is modified to be more suitable for our task. The detailed description of network architecture is provided in Table 2.
The *observation encoder* covers layers 1 to 4, the *sequence encoder* consists in the recurrent layer 5 and the actor-critic network is represented by the two fully-connected layers 6. The observation encoder extracts the features

Table 2
Network architecture

Layer	1	2	3	4	5	6
Parameter	C3x3 32S2	C3x3 32S2	C3x3 32S2	C3x3 32S2	LSTM 256	FC3 FC1

map of input images acquired from the environment; input features are collected over time by an LSTM recurrent layer.

4.3. Advantage actor-critic architecture (A2C)

In the developed RL-A2C learning framework, the observation encoder $f(s_t)$ extracts from the raw images a feature vector ϕ_t which is then used as input for a sequence encoder. The latter is implemented by a LSTM layer as a function $\theta_t = f(\phi_1, \phi_2, ...\phi_t)$ of the observation history $(\phi_1, \phi_2, ...\phi_t)$. After the new feature representation θ_t is extracted from the observation, it is provided as input to both the actor and critic networks. The target of LSTM layer is to organize the process state over time, in this case the information of human position in the image frame acquired by the robot camera sensor. Each time the human moves, the agent needs to collect information to choose a suitable action. Processing the input state as a sequence of images provides much information and speeds up the network update. In this project, the encoder function is re-implemented in the form of the function $h(t) = f_s(h(t-1), \phi_t)$ where $h(t)$ is the hidden state of the recurrent network at time step t. **Reward function** - The reward function used to evaluate the outcome of the actions chosen by the robot during the learning episodes is based on the transform relationship between the robot and human coordinates reference frames (RF). In particular the following parameters are considered:

- α is the relative angle orientation of the human w.r.t. the robot (the angle between the x-axis of the human RF and the x-axis of the robot RF).
- d is the desired distance between the center of the human RF and the center of the robot RF).

The following equation implements the reward function:

$$r = A - B * (d - \sqrt{x_h^2 + y_h^2})^2 - C * |\alpha| \qquad (1)$$

Where A is the maximal reward; B is a scale parameter for the human position w.r.t robot frame; C is a scale parameter for the relative orientation of the human w.r.t robot; x_h and y_h are the position coordinates of the human in the robot RF. The reward shows that if the human is in front of the robot (so fully captured by the camera sensor) and at most two meters far from the robot, the obtained reward is maximized. If the human moves away and changes it orientation w.r.t. the robot, the reward reduces, so the robot has to learn how to move such that the human is always in front of the robot and at the right distance (since also in case the human is too close to the robot, the reward value gets worse).
From the reward function in Equation 1 it is computed the discount reward $R_{t:\infty} = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + ...$, that is used to generate the value function for the actor-critic learning algorithm.

Actor-Critic Networks - In A2C, the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V(s)$ is approximated to a linear function by neural network. The action value function $Q(s, a) = E[R_{t:\infty}|s = s_t, a = a_t, \pi]$ stands for the expected future reward of choosing an action at a particular state, and the value function $V(s) = E[R_{t:\infty}|s = s_t, \pi]$ expresses the value of being in a specific state. The critic network approximates the value function $V(s_t)$, which provides the expected future reward. The actor network outputs the policy distribution $\pi(\cdot; s_t)$ used for action decision. In A2C, $V(s_t)$ and $\pi(\cdot; s_t)$ are used to optimize the network weights during the training process. Thus the value and policy functions are evaluated on the updated model parameters $V(s, \theta')$ and $\pi(a|s_t; \theta')$. Then, the networks can learn by stochastic policy gradient

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log\pi(a_t|s_t; \theta) A(s_t, a_t) + \beta \nabla_\theta H(\pi(a|s_t; \theta)) \qquad (2)$$

$$\theta' \leftarrow \theta' - \alpha \nabla_{\theta'} \frac{1}{2}(R_{t:t+n-1} + \gamma^n V(s_{t+n}; \theta^{'-}) - V(s_t; \theta))^2 \qquad (3)$$

Where $R_{t:t+n-1}$ is the discounted reward for n steps; θ is the discount factor; H is an entropy regularization factor; β is the networks regularization factor; $\theta^{'-}$ is the model parameters set in the previous time step.

5. Training process

The training is performed from scratch, without exploiting pre-trained model weights. In the training process, the human moving is set up to follow the trajectory showed in figure Figure 6. Two training runs consisting of 1000 and 2000 epochs respectively are performed. The Adam optimizer for the training process is used to speed up and stabilize convergence to the global minimum in the gradient descent phase. For further details about hyper-parameters refer to Table 3.

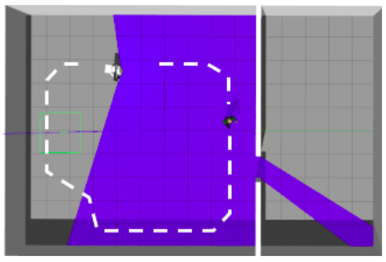


Figure 6: Training trajectory

Table 3
Higher parameters

Train arguments	Value	Reward parameters	Value
Learning rate	1e-4	A	10
Max steps episode	150	B	5
Discount factor	0.99	C	10
Regularizer factor	0.01	d	2
Value loss coefficient	0.5		

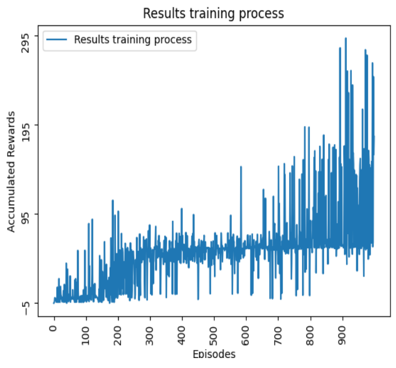
5.1. Training results

During the training process, the accumulated reward per episode is collected. Figure 7 shows the trend of total reward increasing during the training epochs. This demonstrates that A2C framework learned well how to select robot actions to accomplish the considered visual object tracking task. At the end of the training process, the robot can follow the human along the whole trajectory.

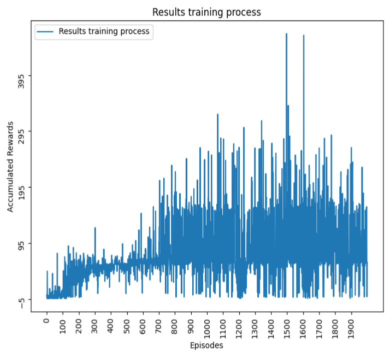
Table 4
Training time

Number episodes	Training time
1000	9 hours
2000	27 hours

One of the main improvements in the training process achieved in this project is the number of episodes required to get the cumulative reward converging to high values, that is smaller with respect to the number of epochs required to the training of RL algorithms for analogous tasks. This is mainly due to the use of the robot model, instead of the camera view, in the RL process, that is more suitable for robotics applications, and to have set the empty room to make the learning process converging quickly. But our training process also has disadvantage that is the time of training. There are two main reasons for that: in Gazebo simulator is not easy to change the time speed as in game engines (frequently used as simulation environments for RL applications) and the low computational resources available.



(a) Training results with 1000 episodes



(b) Training results with 2000 episodes

Figure 7: Accumulated rewards value during training episodes, (a) 1000 epochs training; (b) 2000 epochs training

In general, the algorithm works well, and we think that the developed system can be easily and effectively adapted for real world robots because of the use of robot APIs for the train and test processes.

6. Testing process

In this section it is described the testing framework used to get quantitative and qualitative evaluations of the training process. The test process is articulated as follows:

- Three testing trajectories are defined to analyze and evaluate the performances of the trained model to track the human target in the same indoor environment used for training.
- A new 3D simulation-world is created to test the tracking performances of the trained model when the robot has to move in an indoor environment showing structural and textural differences with respect to the training one.

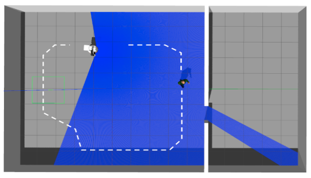
- Different values of robot speed, trajectory (path and duration) for single action steps, reward function parameters are also evaluated during test process.

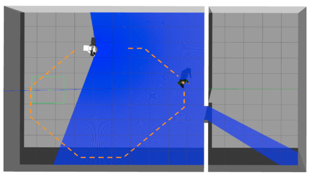
6.1. Trajectories settings

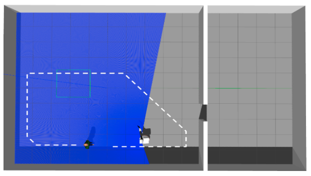
The trajectories produced for the test process are designed to show an increasing level of diversity with respect to the trajectory defined for the training process, which in turn also determines a gradual increase in the complexity of the active tracking task to be learned by the agent. Below a schematic view of the designed trajectories and an overview of the main features are presented:
Base squared trajectory - A first evaluation of the experimental results is performed on the same trajectory used in the training process, aiming to validate the learning results achieved in the training procedure. In this case, the difference with respect to the training trajectory is determined by changes made in the set-up of agent parameters (speed, action steps) and environment parameters (speed of human targets, obstacles, light conditions). See Figure 8(a).

Squared rotated and reshaped trajectory A second level of evaluation is executed on a trajectory which start showing changes in the shape and in the orientation of the path with respect to the environment coordinate frame. Also in this case a different set-up of robot and world parameters is exploited to prove the robustness of the model learning state. In particular, setting different human target speed and robot action steps showed an improvement in evaluation results. Refer to Figure 8(b).

Short and trapezoidal trajectory A further step in the evaluation process is performed on a *new* trajectory which shows no correlation with the training one. The starting and final position of the agent is changed, the shape and orientation of the path in the world frame are different.
The main features of the trajectory set-up are the length of the path that is less than the previous two test cases also due to a very acute angle of the first path curve that enhance the complexity of the visual tracking task.
Suitable settings of the agent action steps and of the human speed produced good results for this test case. See Figure 8(c) **Long eight-shaped trajectory** Finally, the task complexity is further increased defining a test case on a *new* long trajectory characterized by: different start and end position of the agent, complex shape with more curvatures with acute and wider angles, orientation and shaped changed with respect to the trajectory used in all the previous test cases. Also in this case it has been possible to define a suitable set-up for the agent and environment parameters allowing the model to successfully track most of the human target motion along the whole trajectory. Refer to Figure 8(d)


(a) Base squared trajectory


(b) Rotated & reshaped squared trajectory


(c) New short & trapezoidal trajectory

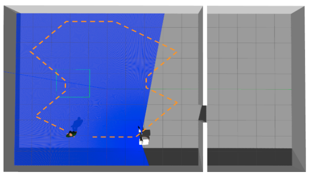
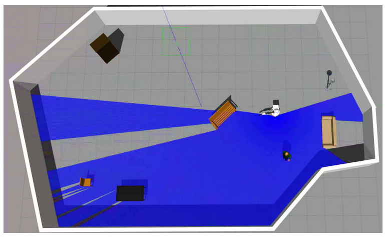

(d) New long & height-shaped trajectory

Figure 8: Trajectories used for test process

6.2. New test 3D simulation world model

To better analyse and validate experimental results, a new simulation indoor world environment is created. The new space is slightly wider in terms of squared meters area, with respect to the world model used for training and for the test study cases described in paragraph 6.1. Moreover, it has a different shapes, with a more irregular intersection of the perimeter walls and also the presence of some obstacles which can considerably affect the noisy of the input images processed by the neural model and obstruct the robot motion. Due to the strong differences in the world setting and also the environmental *simplifications* adopted in training process to speed up learning convergence, the experimental results achieved in this test case are not as concrete as for the results obtained using the original world simulation set-up, but they allow to show anyway some interesting generalization properties acquired by the neural network model in the policy learning process. The new experimental environment and the related test trajectory are shown in Figure 9


(a) new (complex) simulation world environment

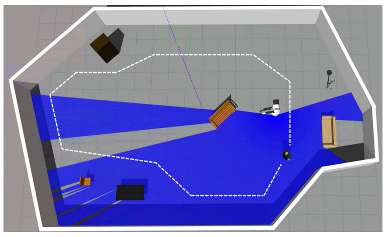

(b) trajectory performed by the actor (human) model

Figure 9: Large (new) irregular-shaped room with obstacles (a) and related test trajectory for the actor model (b)

6.3. Evaluation Metrics

Before analyzing the test results obtained for the considered visual tracking task, it is worth to introduce the performance metrics used for the quantitative evaluation.

- **Accumulated Reward**: sum over the total number of episodes of the reward computed for each action step. Note that the immediate reward defined in Equation (1) measures the goodness of tracking at some time step, so the metric Accumulated Reward is conceptually much like *Precision* in the conventional tracking literature. [1]
- **Episode Length**: measures the duration of good tracking, which shares the same spirit as the *Successfully Tracked Frames* [2] used in conventional tracking applications.
- **Failure Rate**: number of fails (end-of-episode) over the total number of episodes. This measure is analogous to the homonyms metric used for evaluation in the conventional tracking literature.

6.4. Experimental Results

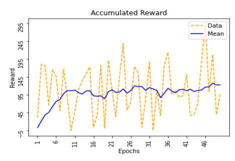
Squared empty room environment – In this section a qualitative evaluation is provided for the test scenarios defined in 6.1 and 6.2. The following plots ad tables show the accumulated rewards, the episode length and the failure rate performance metrics for the testing processes performed on both the typology of simulation environment As for the test results achieved on the base (training) trajectory (Figure 10) one can see that the accumulated reward values are quite high for the whole test run, with peaks reaching a cumulative reward value equal to 295. Also from the plot concerning the episode length metric one can notice from the early epochs the robot can follow the most and, in many cases, the whole trajectory performed by the human, thus proving the goodness and the stability of the training results. The results obtained on the *rotated and reshaped* trajectory (Figure 11) show a little decay of the model performance, considering the mean value of the computed metrics, but looking at the absolute values for the single episodes, we can ascertain that the robot can still follow the entire trajectory, as proved by the high cumulative reward values collected in different episodes, but with a slightly increment of failed episodes. Very similar positive results are also achieved for the *short trapezoidal* trajectory (Figure 12), for which they are still reached high cumulative reward peaks and also the mean cumulative reward along the episode runs and the mean episodes length testify the ability of the model to adapt also to a trajectory with different shape and turning directions with respect to the training one. Finally, the performance indices evaluated on the *long eight-shaped* trajectory (Figure 13) prove how the trained model also scales and generalize quite well on a completely different and more complex trajectory. If the mean cumulative reward is the lowest with respect to the previous test cases, they are still reaching high absolute cumulative values in many episodes run and also the mean episode length value is close to that of the previous (simpler) test cases.

Table 5
Total test runs and failure rates for the four test cases on (training) squared empty world environment

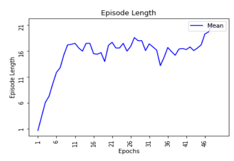
Test environment	Total episodes	Failure rate
Base (training)	50	0.28
Rotated & reshaped	50	0.38
Short & trapezoidal	50	0.46
Long & eight-shaped	50	0.51

[1] In single-target short-term tracking Precision is defined as the average overlap, i.e., the average value in the sequence overlap between predicted target's region form the tracker and the ground-truth region.

[2] This measure reports the number of successfully tracked frames from the tracker's initialization to its (first) failure. The choice of the failure criterion may impact the evaluation result.

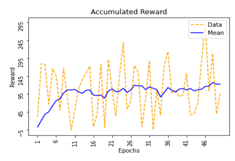


(a) Cumulative-per-episode rewards (orange) & mean cumulative reward over the whole test run (blue)

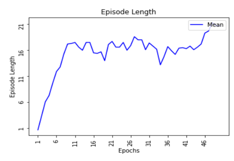


(b) Average episodes lengths over the whole test run

Figure 10: Evaluation results on base squared trajectory

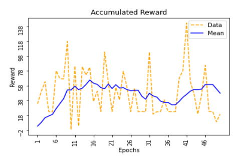


(a) Cumulative-per-episode rewards (orange) & mean cumulative reward over the whole test run (blue)

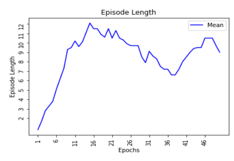


(b) Average episodes lengths over the whole test run

Figure 11: Evaluation results on rotated & reshaped base trajectory

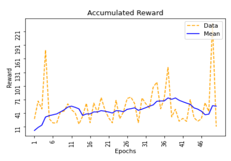


(a) Cumulative-per-episode rewards (orange) & mean cumulative reward over the whole test run (blue)

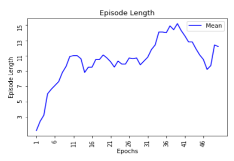


(b) Average episodes lengths over the whole test run

Figure 12: Evaluation results on new short & trapezoidal trajectory



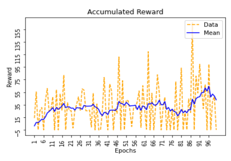
(a) Cumulative-per-episode rewards (orange) & mean cumulative reward over the whole test run (blue)



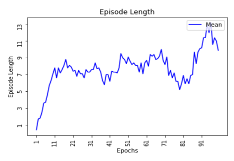
(b) Average episodes lengths over the whole test run

Figure 13: Evaluation results on new long & eight-shaped trajectory

Irregular-shaped with obstacles room environment
In the end, a further evaluation case is defined, to test the generalization capabilities of the trained model and the possibility to extend it also to a *new environment* with a higher degree of complexity (characterized by an irregular perimeter shape and the presence of obstacles). To let the model process in a good way the input (noisier due to the obstacles appearing in the camera frames and the larger field of view required in the considered wider space, we run 100 test episodes, during which both the mean length of the test run and the cumulative reward gradually increase, reaching peak values comparable with the ones obtained for the much simpler test cases considered in 6.1. In particular the robot was able to track and follow the human motion for more than a half of the total trajectory while also avoiding the obstacles displaced in the traversed regions of the room (Figure 14) The strong performance achieved in tracking and obstacle avoidance task are further proved by the relatively low failure rate of the test runs on the new environment, as can be observed by comparing the values in Table 5 and Table 6



(a) Cumulative-per-episode rewards (orange) & mean cumulative reward over the whole test run (blue)



(b) Average episodes lengths over the whole test run

Figure 14: Evaluation results on new irregular-shaped room with obstacles

Table 6
Total test runs and failure rates for the test case on new irregular shaped with obstacles world environment

Test environment	Total episodes	Failure rate
Irregular-shaped with obstacles	100	0.59

7. Conclusions

In this project, we provided an improvement in visual object tracking problem for robotics application. Our aim was to accomplish the task of person following for a mobile manipulator in an indoor environment for healthcare purposes. We modeled the learning task as a Markov Decision Process and exploited the use of robot APIs, of the simulation engine Gazebo, and of the Robot Operative System framework, to set up and deploy the training and testing processes for the Advantage Actor-Critic reinforcement learning algorithm used to make the robot learning the desired task. Our approaches got significant results as shown by the performance evaluation performed in different testing environments. In the future, we aim to deploy this project in real TIAGo robot platform to test the implemented algorithm in real environments.

References

[1] G. Lo Sciuto, S. Russo, C. Napoli, A cloud-based flexible solution for psychometric tests validation, administration and evaluation, volume 2468, 2019, p. 16 – 21.

[2] S. Illari, S. Russo, R. Avanzato, C. Napoli, A cloud-oriented architecture for the remote assessment and follow-up of hospitalized patients, volume 2694, 2020, p. 29 – 35.

[3] S. Russo, C. Napoli, A comprehensive solution for psychological treatment and therapeutic path planning based on knowledge base and expertise sharing, volume 2472, 2019, p. 41 – 47.

[4] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, volume 2768, 2020, p. 41 – 45.

[5] M. Wozniak, C. Napoli, E. Tramontana, G. Capizzi, G. Lo Sciuto, R. Nowicki, J. Starczewski, A multiscale image compressor with rbfnn and discrete wavelet decomposition, volume 2015-September, 2015. doi:10.1109/IJCNN.2015.7280461.

[6] R. Brociek, G. De Magistris, F. Cardia, F. Coppa, S. Russo, Contagion prevention of covid-19 by means of touch detection for retail stores, volume 3092, 2021, p. 89 – 94.

[7] B. Torkaman, M. Farrokhi, Real-time visual tracking of a moving object using pan and tilt platform: A kalman filter approach, in: 20th Iranian Conference on Electrical Engineering (ICEE2012), 2012, pp. 928–933. doi:10.1109/IranianCEE.2012.6292486.

[8] Y. Çelik, M. Altun, M. Güneş, Color based moving object tracking with an active camera using motion information, in: 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 2017, pp. 1–4. doi:10.1109/IDAP.2017.8090332.

[9] I. Kostrikov, Pytorch implementations of asynchronous advantage actor critic, https://github.com/ikostrikov/pytorch-a3c, 2018.

[10] Y. Zhu, R. Mottaghia, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, Target-driven visual navigation in indoor scenes using deep reinforcement learning (2016). URL: https://arxiv.org/abs/1609.05143. doi:10.48550/arXiv.1609.05143. arXiv:1609.05143.

[11] N. Brandizzi, V. Bianco, G. Castro, S. Russo, A. Wajda, Automatic rgb inference based on facial emotion recognition, volume 3092, 2021, p. 66 – 74.

[12] L. Tai, G. Paolo, M. Liu, Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation (2017). URL: https://arxiv.org/abs/1703.00420. doi:10.48550/arXiv.1703.00420. arXiv:1703.00420.

[13] J. Choi, J. Kwon, K. M. Lee, Real-time visual tracking by deep reinforced decision making (2018). URL: https://arxiv.org/abs/1702.06291. doi:10.48550/arXiv.1702.06291. arXiv:1702.06291.

[14] X. Ye, Z. Lin, H. Li, S. Zheng, Y. Yang, Active object perceiver: Recognition-guided policy learning for object searching on mobile robots (2018). URL: https://arxiv.org/abs/1807.11174. doi:10.48550/arXiv.1807.11174. arXiv:1807.11174.

[15] O. Robotics, rocker, https://github.com/osrf/rocker, 2018.

[16] V. P. S. Bharat Singh, Rajesh Kumar, Reinforcement learning in robotic applications:a comprehensive survey, IEEE Transactions on Neural Networks and Learning Systems (2021). doi:10.1007/s10462-021-09997-9.

[17] V. Konda, J. Tsitsiklis, Actor-critic algorithms, in: S. Solla, T. Leen, K. Müller (Eds.), Advances in Neural Information Processing Systems, volume 12, MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, CoRR abs/1602.01783 (2016). URL: http://arxiv.org/abs/1602.01783. arXiv:1602.01783.

[19] W. Luo, P. Sun, Y. Mu, W. Liu, End-to-end active object tracking via reinforcement learning, CoRR abs/1705.10561 (2017). URL: http://arxiv.org/abs/1705.10561. arXiv:1705.10561.