Università di Pisa

Scuola di Ingegneria

Corso di Laurea Magistrale in Ingegneria Robotica e dell'Automazione

Tesi di Laurea Magistrale

Development and Testing of a Software Framework for Controlling Humanoid Robots in Disaster-Response Scenarios

Relatore:
Prof. Antonio BICCHI
Controrelatore:
Prof. Carlo Alberto AVIZZANO

Candidato:
Enrico CORVAGLIA

Anno Accademico 2015-2016

"Success is the ability to go from failure to failure without losing your enthusiasm."
Winston Churchill

Abstract

The aim of this thesis is to design and develop a modular software framework for controlling humanoid robots in teleoperation, in a context of disaster-response or civil defense. Over the years, natural (earthquakes, floods, etc.) or man-made disasters (nuclear reactor meltdowns, terrorist attacks, etc.) have caused several victims. The state of the art of disaster-robotics allows to deploy efficient and powerful robots in order to assist and support humans in the delicate phases of searching and rescuing survivors. In particular, with the use of teleoperation, the inclusion of a human operator (human-in-the-loop) can dramatically promote the application of humanoid robots, due to the human superior competence in critical thinking and context-awareness. This way, robots can be used as an interface between man and environment. Under these concepts, the thesis work focused on the design of a robust and efficient control architecture that brings whole-body locomotion and manipulation capabilities to the robot. Specifically, this thesis dealt with the development of a software module for teleoperating a robot while it is in a vehicle, making it able to drive. The module internal architecture is structured as a Finite State Machine, which allows to model a workflow of behaviors in an event-driven manner, providing safe and robust control in a teleoperation scenario. The effectiveness of the developed software has been validated during the *DARPA Robotics Challenge Finals*, occured in Pomona, CA (USA), on June 5-6 of 2015.

Contents

List of Figures

Chapter 1

Introduction

1.1 Disaster Robotics

In recent years, robotics is going through a solid revolution in terms of development and research. More often, the role of humans in the majority of the manufacturing processes is being substituted by robots or, in general, automated machines. This implies not only an improvement of productivity and time management, but it adds also the possibility for the human beings to avoid risky and dangerous operations. Extending these concepts outside the industries, in general in unstructered environments, robots could supply a remarkable support in terms of defence and human aid. In particular, a whole branch of robotics is heading towards the so-called *disaster robotics* [1]. Unfortunately, over the years, natural disasters (earthquakes, floods, etc.) and man-made disasters (nuclear reactor meltdowns, terrorist attacks, etc.) have caused victims not only during the accident itself, but also during the search and rescue phase (Figure 1.1). The aim of disaster robotics is exactly to employ efficiently designed robots to operate in the delicate and dangerous tasks of rescuing accident survivors, providing a rapid emergency-response to limit the fallout. A variety of robot designs exists depending on the situation and the specific task, such as wheeled, snake-like or humanoid robots. While wheeled robots could result more agile on rough surfaces, humanoids are more versatile and powerful, allowing motions and employments similar to human ones, in particular complex manipulation tasks like moving rubbles away, using advanced tools, opening doors, driving vehicles and also bringing in-loco medical assistance. This thesis will take account of the humanoid robots category.

Figure 1.1: Top: Fukushima nuclear reactor explosion. Bottom: human workers exposed to radiations during the emergency-response phase.

1.1.1 The Need of Teleoperation

Despite the current evolution in the robotics field has brought great improvements and capabilities to humanoid robots, their state of the art technology is still limited in terms of autonomy. This limitation can be perceived even more in disaster scenarios, where the readiness of decision making is crucial. It is thus clear that the human competence in critical thinking and context-awareness must be taken into account. With the use of teleoperation, the inclusion of a human operator (human-in-the-loop) can dramatically promote the application of humanoid robots. This perspective motivates teleoperation as a primary need in disaster-response scenarios, because robots are deficient in many behaviors with respect to humans [2]. There are also other positive motivations for using teleoperation, not directly related to robots tecnhology, which include some ethical consideration. Since this branch of robotics is pointing towards direct human-robot interactions, such as rescuing a wounded person, a human operator could be able to instill moral accountability and respect for the human life, in robots.

Figure 1.2: Illustration of humanoid robots employed in a disaster-response scenario.

1.2 DARPA Robotics Challenge

The Fukushima Dai-ichi nuclear disaster, occured in March 2011, represented a turning point for the robotics community. The disaster involved three explosions in the nuclear reactor, which, according to a following analysis, could have been prevented if there had been a proper intervention in the very first hours after the accident. Although the American governative agency *DARPA* provided the robots for supporting the rescue operations and site remediation, they were good only for inspection, while lacking some essential functions such as turning a valve or cutting openings in the wall. According to DARPA, this situation highlighted the need to focus on the development of robots that can perform complex tasks in hazardous environment (Figure 1.2). For these reasons, DARPA announced a competition, the DRC, with the aim of pushing forward the state of the art of disaster robotics [3]. Quoting the DRC Program Manager, Dr. Gill Pratt:

> *The DRC is a competition of robot systems and software teams vying to develop robots capable of assisting humans in responding to natural and man-made disasters. It was designed to be extremely difficult. Participating teams, representing some of the most advanced robotics research and development organizations in the world, are collaborating and innovating on a very short timeline to develop the hardware, software, sensors, and human-machine control interfaces that will enable their robots to complete a series of challenge tasks selected by DARPA for their relevance to disaster response.*

As shown in Figure 1.3, robots had to perform different tasks, teleoperated from human operators located far from the robots, in an environment that reconstructs the disaster scenario of Fukushima.

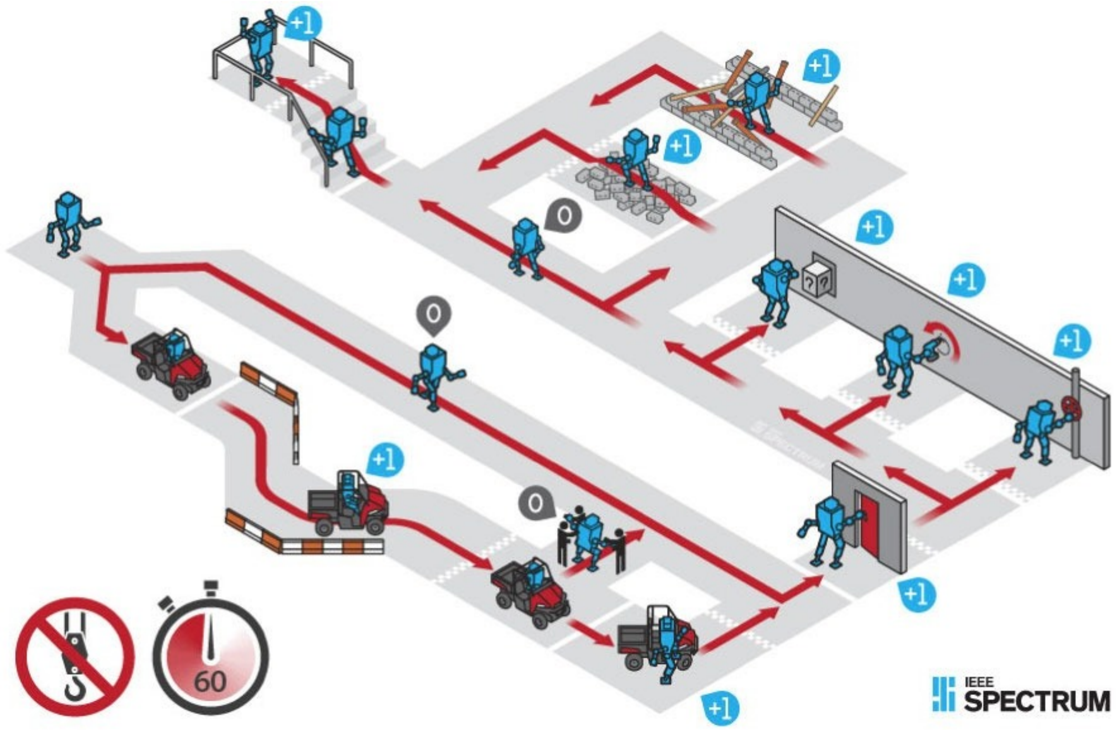


Figure 1.3: DARPA Robotics Challenge Finals circuit.

In particular, the task sequence was:

- *drive* a vehicle from a safe area to a hazardous area;
- *egress* from the vehicle;

- *open a door* to enter a building;
- *open a valve*;
- *cut through a wall* using a tool;
- *traverse* a rough terrain;
- *cross over* rubbles;
- *climb stairs* to exit the hazardous area.

In addition, in order to better simulate a typical disaster scenario, all the communications between robots and operators were performed using a degraded network channel.

The DARPA Robotics Challenge experience involved the best robotics teams from all over the world, showing off some of the most innovative and cutting-edge robotics technologies (Figure 1.4). On the other hand, a number of limitations and drawbacks emerged [4]. A detailed analysis on the DRC event will be discussed on chapter 6.

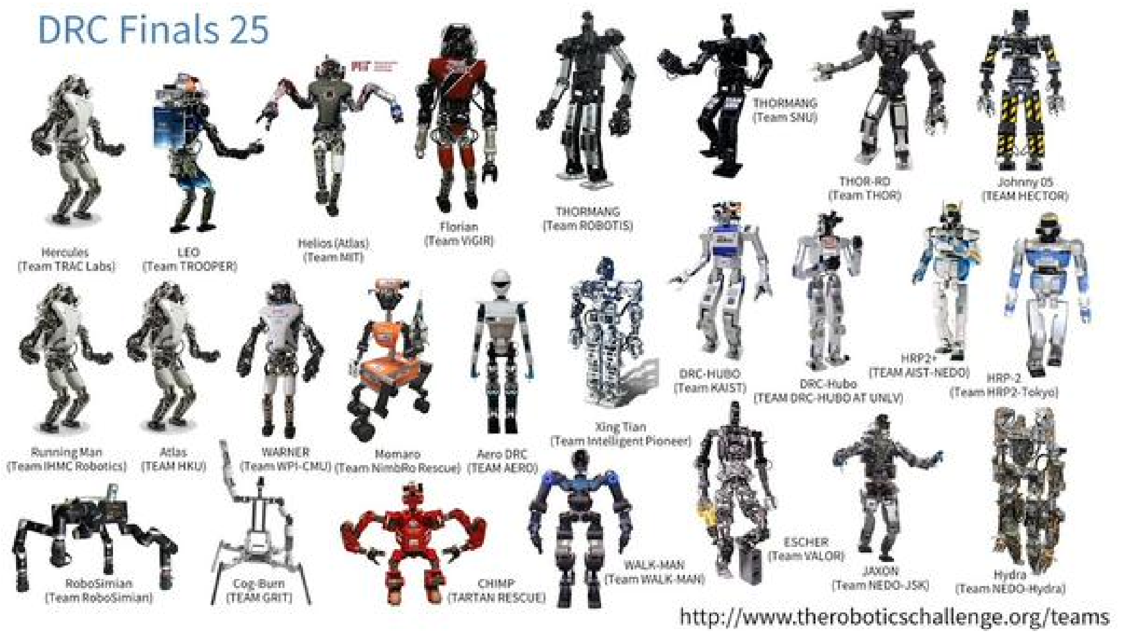


Figure 1.4: DARPA Robotics Challenge Finals participating teams.

1.3 Problem Statement

This thesis treats the development of a software framework for controlling a semi-autonomous humanoid robot, teleoperated from a human operator who makes primary decisions and supervises all its actions. The robot, receiving high-level commands, is capable of elaborating its own decisions through its artificial intelligence, performing the commanded task at its best. This way, the robot acts exactly as an interface between the operator and the environment, which is very convenient in disaster scenarios. Specifically, the thesis work focused on the design of a robust and efficient control algorithm that brings the robot the capability to drive a vehicle. In a disaster scenario, the robot could reach the disaster area and bring in-loco assistance, while leaving human beings in safety.

The thesis is organized as follows. In Chapter 2, the robot platform employed for the development is presented, along with its structure and its main features. In Chapter 3, a background of the multi-layered software architecture is presented, including the *Pilot Interface*, which provides all the tools needed for a teleoperation system. In Chapter 4, the Whole-body control problem is presented, with details on Inverse Kinematics-based algorithms and a novel framework for managing task prioritization with constraints. Chapter 5 focuses on the implementation of a software module, used to enable the robot to perform a driving task in teleoperation. In Chapter 6, the main results from this thesis work, and future developments are presented.

Chapter 2

Robot Platform

2.1 Walk-Man

The robot platform used for this thesis work is *Walk-Man* (Whole-body Adaptive Locomotion and Manipulation), a prototype of an adult-size humanoid robot [5]. Walk-Man is designed and developed under the collaboration between *Università di Pisa* and *Istituto Italiano di Tecnologia* of Genova, with the primary goal of providing support to the Italian Civil Defense in emergency-response scenarios, and in general to provide assistance for dangerous tasks.

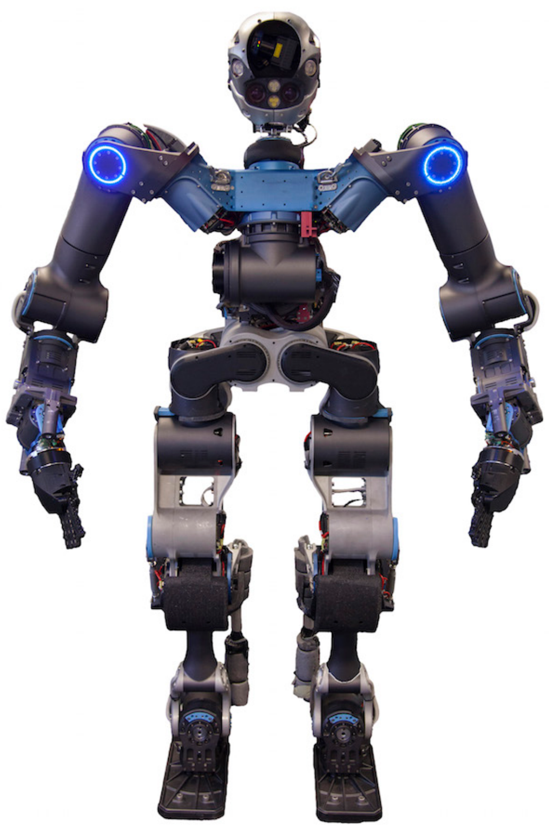


Figure 2.1: The *Walk-Man* robot.

2.1.1 Main Features

Compliance

The concept behind the design of Walk-Man is based on the *soft robotics*. The ever-growing need for robots in social activities, including the interaction with humans, is leading to go beyond the classic assumptions of stiff robotics. The role of soft robotics is thus to increase adaptability and robustness for creating a new generation of robots, in the support of humans within their natural environments. The Walk-Man robot has 33 DOFs actuated by high power electric motors, equipped with intrinsic elasticity (*Series Elastic Actuators*) [6], that allow the robot to gain a compliant behavior and to improve physical interaction capabilities.

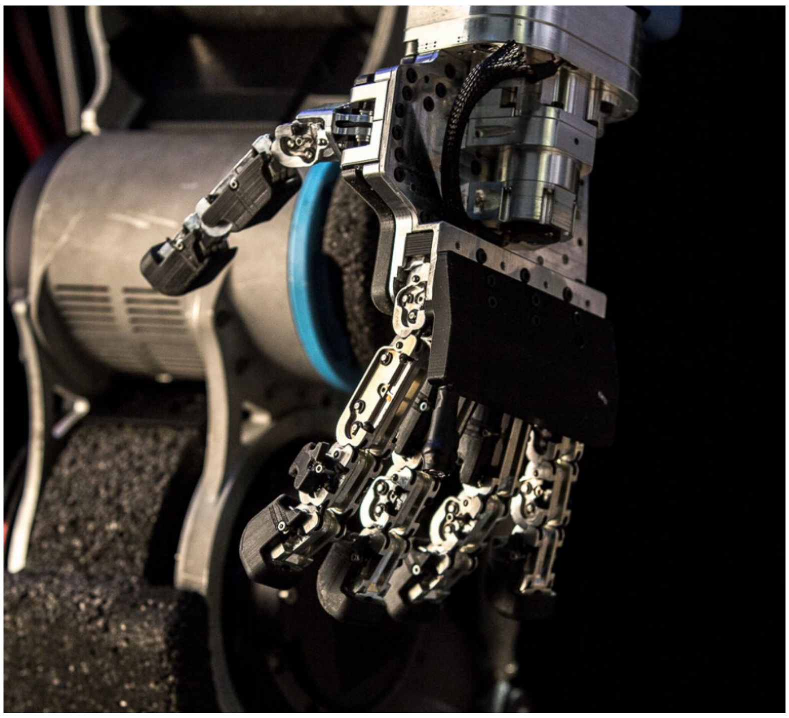


Figure 2.2: The *Pisa/IIT SoftHand*.

Manipulation

Another key component that extends the concept of compliance is the robot hand, which uses the *Pisa/IIT SoftHand* prototype [7]. The Pisa/IIT SoftHand is an extremely under-actuated anthropomorphic hand that has 19 joints, but uses a single motor to control them. The concept behind this kind of under-actuation is that of *synergies*, which is inspired by the coordinated and ordered ensemble of the human hand motion. The major benefit of this design is the great adaptability of the hand, allowing the grasp to be performed even with position/orientation error with respect to the object. In addition, the SoftHand is very robust and it can withstand severe forces exerted on it.

Perception

The perception system has the aim of augmenting the context-awareness capabilities of a robot, and, indirectly, of the human who operates it. The Walk-Man robot perception system is equipped with:

- A 3D range sensor module (*MultiSense SL*), mounted on the robot head and composed by a stereo camera, a rotating 3D laser scanner, and a RGB camera. This module is used to reconstruct the surrounding environment with high-resolution details;
- A set of force/torque sensors, located at the end-effectors, useful to handle the interaction with the enviroment;
- A set of inertial measurement units (IMU), that provides the necessary body orientation sensing.

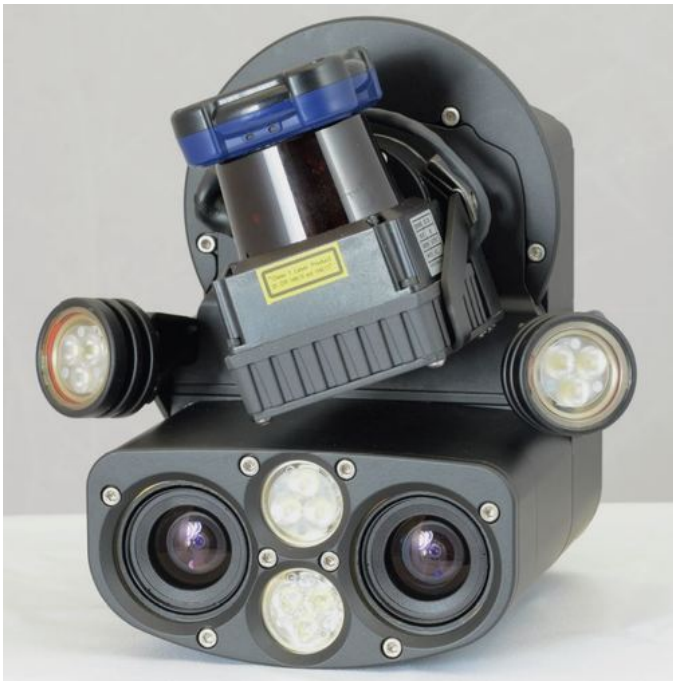


Figure 2.3: The *MultiSense SL* range sensor module.

Chapter 3

Software Architecture

3.1 Semi-autonomous Approach

The state of the art of humanoid robotics is still relatively immature in terms of robots' autonomous capabilities. In addition, especially when dealing with disaster scenarios, issues such as telecommunications degradation could occur, making some kind of human intervention strictly necessary. Hence, the software architecture employed in this work follows a teleoperation-oriented approach, where a human *operator* (also referred to as *the pilot*) has an active role in the control system [8, 9]. This type of approach, also known as *semi-autonomous* approach, allows the operator to focus on high-level control and supervisory tasks, while leaving the robot to address low-level tasks. All of these concepts fall under the field of *Human-Robot Interaction* (HRI).

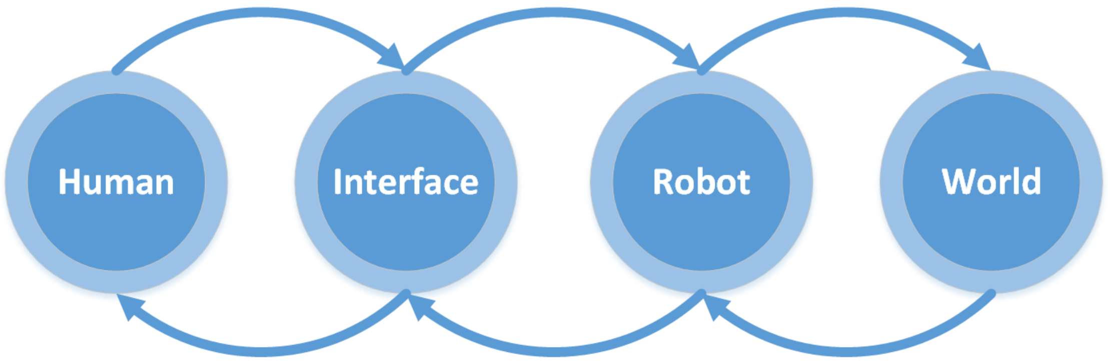


Figure 3.1: Human-robot interaction scheme.

A benefit of the semi-autonomous control approach is that the operator can dynamically set the level of autonomy of the robot, according to the task or the context. In this way, the operator can exploit low-level robot capabilities at their best, when needed, without losing focus on the whole operation. Also, in terms of data exchange between the robot and the operator, this system leads to a remarkable improvement of the bandwith used, which could be crucial in disaster-response scenarios.

The implementation of the multi-level autonomy takes inspiration from the *Motion Description Language* (MDL) approach [10]. MDL is a formal language used to abstract different types of complex behaviors into a set of basic control laws, called *primitives*. In humanoid robotics, where high-DOFs robots are employed, the motion control problem adopts the *operational-space* formulation, where tasks are defined in the cartesian space instead of the robot joint space [11]. In this view, a single cartesian task can be seen as a primitive in the MDL.

Following these principles, it is possibile to characterize different methods for managing robot autonomy in teleoperation-oriented systems:

Traded Control The top-level provides the higher level of autonomy to the robot. The action of the operator is thus limited to initiating a task or a behavior (including the composition of primitives) for a robot to follow, which the robot performs autonomously. Of course, the operator can stop the robot or issue new tasks at any time.

Shared Control In *Shared Control*, the operator continuously provides input to the robot. These inputs, that could be for example points in the cartesian space, are interpreted by the robot, which generates the related behavior in order to accomplish the goals. In addition, the robot has the necessary autonomy to modify operator's inputs if it believes that those actions would violate certain safety objectives, such as colliding with obstacles or losing balance.

Direct Control In *Direct Control*, no robot autonomy is used, since the operator manually controls the robot by regulating the joint displacement.

In general, it is important that the operator could be able to switch seamlessly between the different levels of robot autonomy.

3.2 Software Layers

The software architecture used for this work, as shown in Figure 3.2, is organized into four layers:

- A GUI used to remotely control the robot, the *Pilot Interface*.
- The *control and perception modules* on the robot-side, connected to the operator-side through a *network bridge*.
- A *hardware abstraction layer* (HAL) that remotizes the robot hardware, exposing it to its upper layer.
- The firmware of the actuators running in embedded boards.

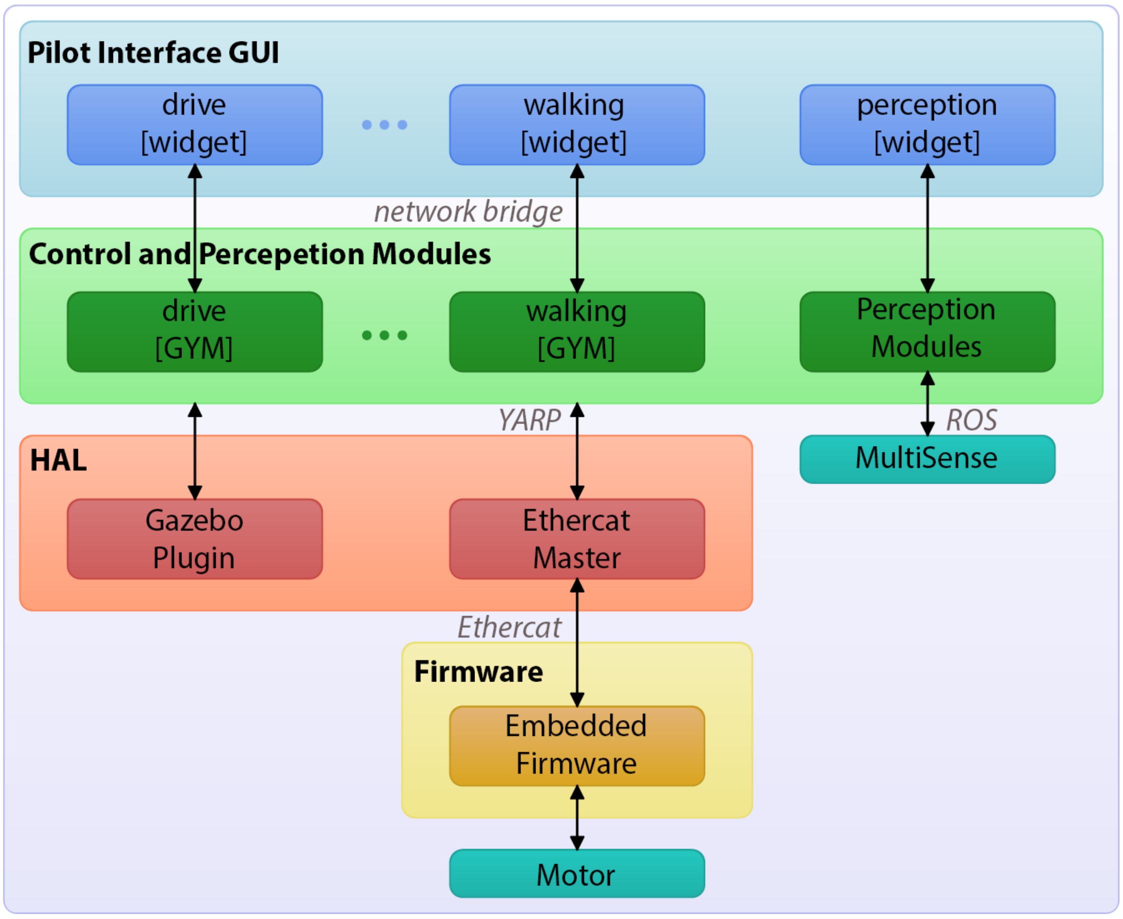


Figure 3.2: Software architecture layers.

Two middlewares supports the whole architecture, in particular:

ROS *Robot Operating System* [12], a community-driven framework, is adopted for the *Pilot Interface* GUI, to visualize the state of the robot within a 3D-reconstructed environment and to handle the perception hardware, including a 3D camera sensor and a LIDAR sensor.

YARP *Yet Another Robot Platform* [13], a robotics framework used for the development of the *control modules* and for handling communications between the robot and the operator (*network bridge*). Also, it provides a set of libraries and tools to manage the *hardware abstraction layer*, useful to interface the above-mentioned modules with the low-level components of the robot.

This section will focus on the first two layers, namely the *control modules* and the *Pilot Interface*.

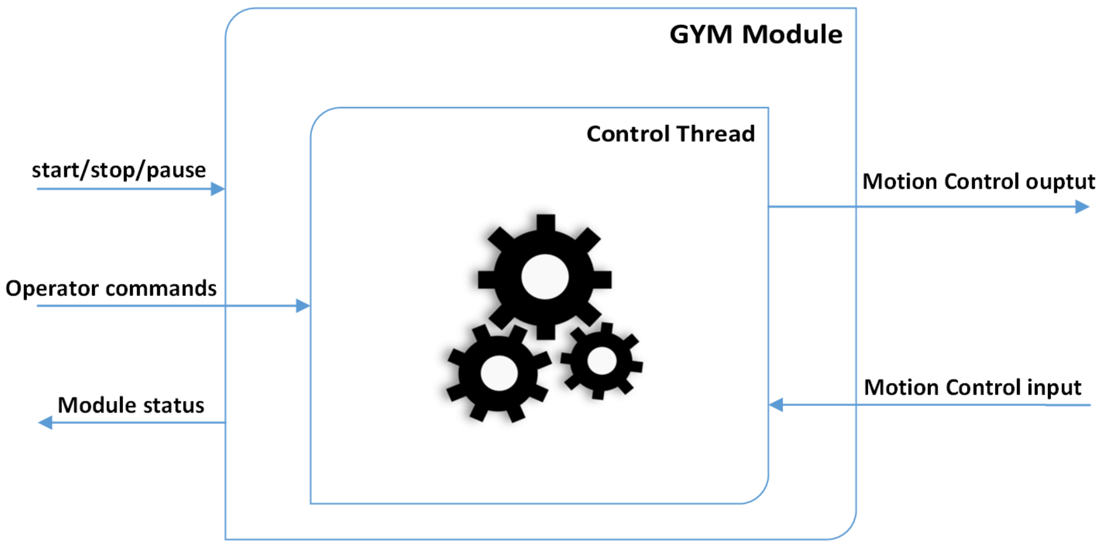
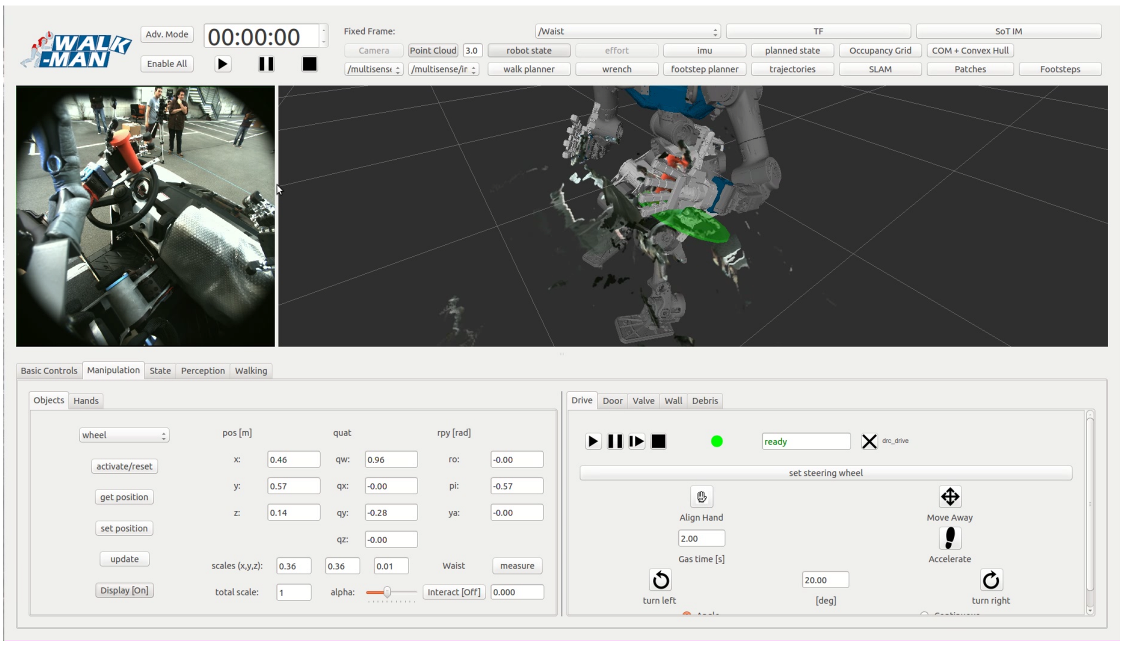


Figure 3.3: Generic YARP Module structure.

3.2.1 Generic Control Module

Based on a modular approach, each module can be considered as a node of a distributed system. In particular, since all *control modules* share a significant portion of code and procedures, it could lead to an inconvenient duplication of code. To avoid this overhead, the software is provided with a generic control module based on YARP, called *Generic YARP Module* (GYM) [14]. A GYM is composed of two threads: a watchdog and a control loop thread (Figure 3.3). The watchdog is a low-frequency thread running at 1 Hz. Its inputs are the commands used to manage the execution of the module, such as *start,stop,pause,resume,quit* (the *Switch Interface*). The output is the status of the module (the *Status Interface*), which also depends on the internal state of the control loop thread. The control loop thread is the main thread of the module, running at higher frequency with respect to the watchdog, in a range between 100 Hz and 500 Hz. It represents the core of GYM, and it implements the robot motion control through a set of primitives, which can be chained to define a complex task, such as driving a car, turning a valve or opening a door [15, 16]. The control loop thread accepts high-level commands as input from the operator (the *Command Interface*) and it uses them to evolve a *Finite State Machine*. In fact, each primitive is implemented within a state of the state machine. After a new state transition, a trajectory is planned for one or more links, which is then sent to an Inverse Kinematics solver. Finally, the solver takes care of the computation of the control law that regulates the robot motion. The detailed description of the components of a GYM module will be explained in chapter 5, where a module designed for the driving task is presented.



00:00:00

Fixed Frame: /Waist

Point Cloud 3.0

/multisense /multisense/ir

wheel

pos [m] x: 0.46 y: 0.57 z: 0.14

quat qw: 0.96 qx: -0.00 qy: -0.28 qz: -0.00

rpy [rad] ro: -0.00 pi: -0.57 ya: -0.00

scales (x,y,z): 0.36 0.36 0.01 Waist

total scale: 1 alpha: 0.000

ready drc_drive

Gas time [s] 2.00

[deg] 20.00

Figure 3.4: The Pilot Interface GUI during a driving task.

3.2.2 Pilot Interface

The *Pilot Interface* is the top-level layer of the software architecture and, as the name suggests, has the purpose of providing a visual interface to remote control a robot (Figure 3.4). It is a user-friendly GUI (*Graphical User Interface*), where the operator could monitor the environment and the robot status and could make correct decisions to perform the tasks. Basically, the Pilot Interface layout is composed of:

- Visualization/Perception panel:
  - a first-person view of the robot from its main camera;
  - an interactive environment, including a third-person view of the robot state and its surrounding scene in the form of point clouds, reconstructed by the perception devices.
- Control panel:
  - a set of task-specific and generic control widgets, in which there are different buttons corresponding to high-level tasks, such as *reach* or *grasp* an object, or *walk* 10 meters straight;
  - a widget that allows to manage a set of 3D-interactive-markers. These markers are useful for referring the pose of some objects of interest in the enviroment, with respect to a known robot frame (Figure 3.5). For example, for a manipulating task like turning a valve, the operator superimposes the corresponding marker over the 3D-reconstructed point cloud of the valve.

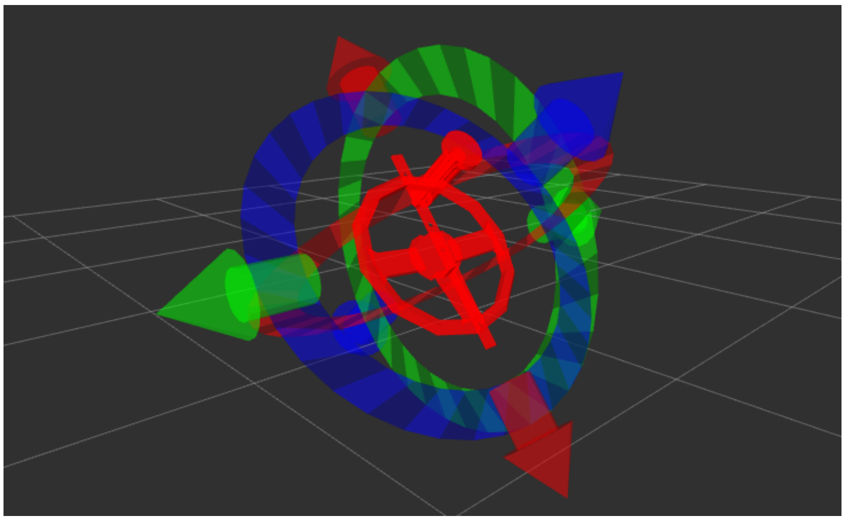


Figure 3.5: 3D Interactive Marker of a steering wheel, used during the driving task.

3.3 Robot Simulation

During the phases of design and development of robotics projects, an invaluable tool is certainly a good simulator. By accurately simulating robots and environments, software designed to operate on a real robot can be executed and validated on the equivalent simulated system. For this thesis work, the open-source Gazebo simulator is used [17], along with a set of plugins that allows the simulated robot to communicate with the YARP framework [18]. Since these plugins share the same interfaces of the hardware abstraction layer, they enable the interoperability of the YARP modules between a real robot and its simulated counterpart. For instance, in Figure 3.6 are shown some of the tests performed during the development of the driving task module.

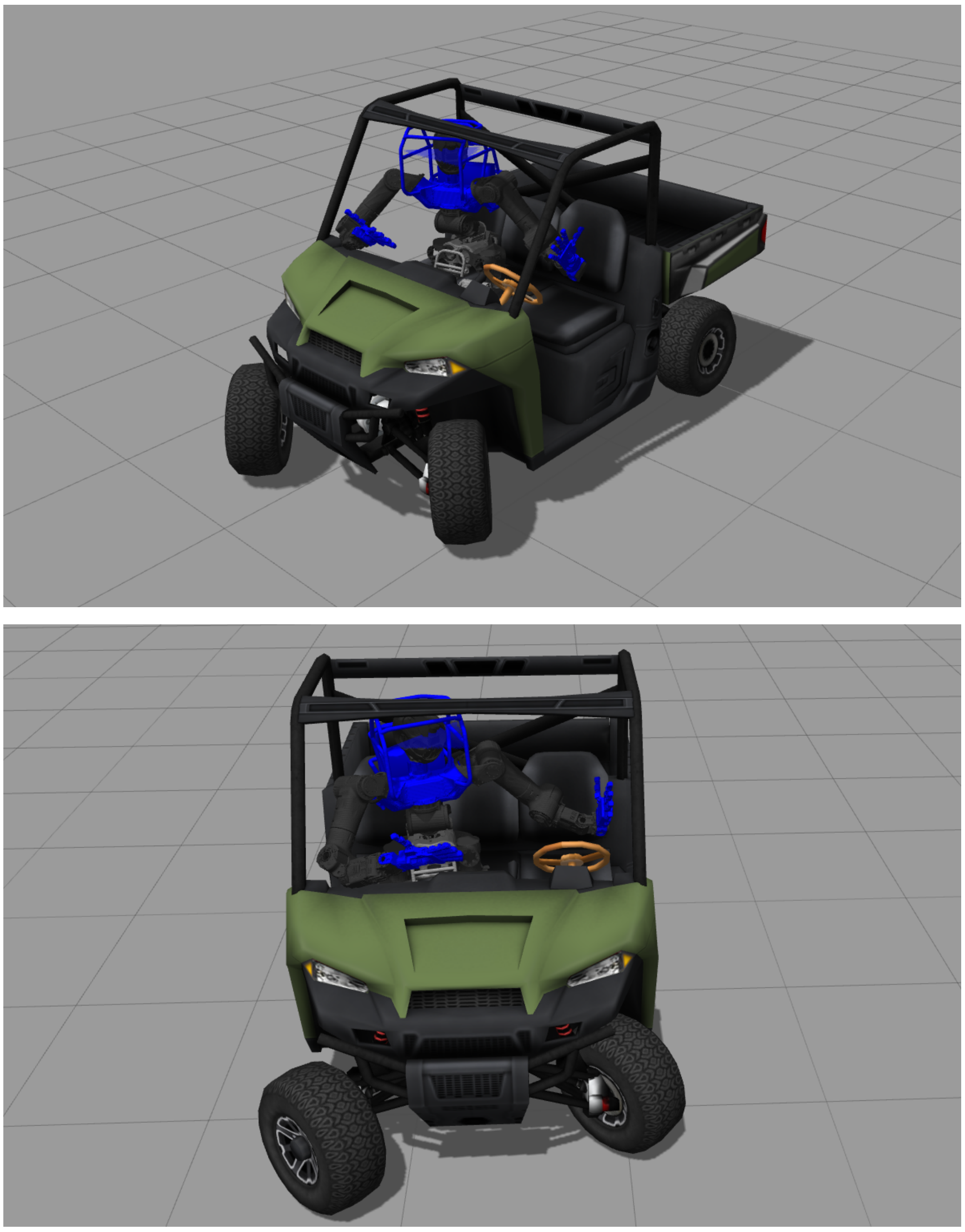


Figure 3.6: Example of Walk-Man robot driving in a simulated environment.

Chapter 4

Whole-Body Control

Nowadays robots are becoming increasingly efficient in performing many different and complex tasks, such as manipulating objects, running or jumping. But in real world applications, a critical limitation of robots is that, in most cases, they are used to perform these tasks individually. To address these constraints, a promising reasearch field pointing towards the right direction is *Whole-Body Control*. Whole-Body Control takes inspiration from the human behaviour of exploiting full capabilities of the entire body when addressing different tasks simultaneously. This control framework finds great applications in highly redundant robots, such as humanoid robots, for executing multiple tasks at different levels of priority, even including the interaction with the environment.

4.1 Inverse Kinematics

A fundamental problem in robotics, which goes under the name of *Inverse Kinematics* problem, is that of mapping tasks or objectives defined in the *operational space*, into the corresponding *joint-space* commands. Thus, given a desired task-space end-effector pose, the Inverse Kinematics problem consists in determining the corresponding joint configuration. Since the problem is in general non-linear, it is not possible to compute a solution in closed-form. This can lead to different scenarios, i.e. no admissible solutions (due to robot's kinematic structure, such as joint limits), multiple solutions or infinite solutions. The latter case occurs when the robot is *redundant*, which means that the robot's degrees of freedom n are greater than the desired task dimension m.

Deriving with respect to the time the following relationship

$$\mathbf{x} = \mathbf{f}(\mathbf{q}) \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^m$ represents the operational-space pose, function of joint variables $\mathbf{q} \in \mathbb{R}^n$, leads to:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{4.2}$$

where the Jacobian matrix $\mathbf{J} \in \mathbb{R}^{m \times n}$ is the linear operator that maps $\dot{\mathbf{q}}$, the joint velocities, into $\dot{\mathbf{x}}$, the operational-space velocities of a distal link, usually an end-effector. For a redundant robot, a general solution for the *Differential Inverse Kinematics* problem, which is the inverse of the problem (4.2), is:

$$\dot{\mathbf{q}}_d = \mathbf{J}^{\dagger}\dot{\mathbf{x}}_d + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\dot{\mathbf{q}}_0 \tag{4.3}$$

where $\mathbf{J}^{\dagger}$ is the Jacobian pseudo-inverse and $\dot{\mathbf{q}}_0$ is an arbitrary joint-space vector of velocities. Considering $\dot{\mathbf{q}}_0$ as a solution of a lower priority task, as in:

$$\dot{\mathbf{q}}_0 = \mathbf{J}_0^{\dagger}\dot{\mathbf{x}}_0 \tag{4.4}$$

it is possibile to introduce a *task prioritization* by projecting $\dot{\mathbf{q}}_0$ onto the null space of the first task's Jacobian $\mathbf{J}$ through the projection operator $(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})$. The lower priority task $\dot{\mathbf{q}}_0$ is executed at its best without interfering with the higher priority task $\dot{\mathbf{q}}_d$. A fundamental framework that extends the task prioritization problem ideally to n-tasks is referred as *Stack of Tasks* (SoT) [19].

4.2 Hierarchical Inverse Kinematics: the *Stack of Tasks*

Considering a redundant robot that has to execute n tasks simultaneously, each task T_i can be described by its corresponding task error $\mathbf{e}_i$. The time derivative of the error $\mathbf{e}_i$ is expressed as:

$$\dot{\mathbf{e}}_i = \frac{\partial \mathbf{e}_i}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \mathbf{e}_i}{\partial t} = \mathbf{J}_i\dot{\mathbf{q}} + \frac{\partial \mathbf{e}_i}{\partial t} \tag{4.5}$$

where $\mathbf{J}_i$ is the Jacobian associated with the i^{th} task. To ensure error convergence, the error dynamics is constrained to follow exponential dynamics:

$$\dot{\mathbf{e}}_i = -\lambda \mathbf{e}_i \tag{4.6}$$

Combining (4.5) with (4.6), leads to

$$\mathbf{J}_i \dot{\mathbf{q}} = -\lambda \mathbf{e}_i - \frac{\partial \mathbf{e}_i}{\partial t} = \dot{\mathbf{e}}_i^* \tag{4.7}$$

Hence, a set of tasks can be fully described by $T_i = (\mathbf{J}_i, \dot{\mathbf{e}}_i^*)$. A typical limitation of the simple Inverse Kinematics control scheme is that it doesn't take into account constraints in the form of inequalities. Thus, the Inverse Kinematics problem can be integrated into a *Quadratic Programming* (QP) problem with linear constraints. A solution for a generic task can be computed solving the following QP problem

$$\begin{aligned} \dot{\mathbf{q}}_i = \underset{\dot{\mathbf{q}}}{\operatorname{argmin}} \, & \|\mathbf{J}_i \dot{\mathbf{q}} - \dot{\mathbf{e}}_i^*\| \\ s.t. \quad & \mathbf{A}_{c,i} \dot{\mathbf{q}} \leq \mathbf{b}_{c,i} \end{aligned} \tag{4.8}$$

The formulation used in (4.8) allows also to express lower and upper bounds for the constrained variables, as well as to use equality constraints. In general, for the n^{th} task, the problem can be written as a cascade of QP problems:

$$\begin{aligned} \dot{\mathbf{q}}_d = \quad \underset{\dot{\mathbf{q}}}{\operatorname{argmin}} \quad & \|\mathbf{J}_n \dot{\mathbf{q}} - \dot{\mathbf{e}}_n^*\| \\ s.t. \quad & \mathbf{A}_1 \dot{\mathbf{q}} = \mathbf{A}_1 \dot{\mathbf{q}}_1 \\ & \vdots \\ & \mathbf{A}_{n-1} \dot{\mathbf{q}} = \mathbf{A}_{n-1} \dot{\mathbf{q}}_{n-1} \\ & \mathbf{A}_{c,1} \dot{\mathbf{q}} \leq \mathbf{b}_{c,1} \\ & \vdots \\ & \mathbf{A}_{c,n} \dot{\mathbf{q}} \leq \mathbf{b}_{c,n} \end{aligned} \tag{4.9}$$

where $\dot{\mathbf{q}}_d$ is the desired joint velocities vector, while the equality constraints guarantee the correct hierarchy between tasks. The solution vector $\dot{\mathbf{q}}_d$ is then usually integrated and used as control input for the robot:

$$\mathbf{q}_d = \mathbf{q} + \dot{\mathbf{q}}_d \Delta t \tag{4.10}$$

where Δt is the control loop sample time.

4.2.1 OpenSoT

A novel Whole-Body Inverse Kinematics oriented tool, *OpenSoT* [20], is used as high level robotics library to easily describe the control problem, including the definition of tasks, constraints and bounds. The main structures of interest of this library are:

Cartesian Task

A *Cartesian Task* can be defined as a composition of a Cartesian position task and a Cartesian orientation task. At each step, OpenSoT computes the Jacobian ${}^{b}\mathbf{J}_d$, according to given distal link and base link. In addition, it computes the Cartesian errors, in position and orientation:

$$\begin{aligned} \mathbf{e}_p =& \quad \mathbf{p}_d - \mathbf{p} \\ \mathbf{e}_o =& \quad -(\eta_d \boldsymbol{\epsilon} - \eta \boldsymbol{\epsilon}_d + [\boldsymbol{\epsilon}_d \times] \boldsymbol{\epsilon}) \end{aligned} \tag{4.11}$$

where $\mathbf{p}_d = \begin{bmatrix} x_d & y_d & z_d \end{bmatrix}^T$ and $\boldsymbol{\alpha}_d = \begin{bmatrix} \eta_d & \epsilon_{1,d} & \epsilon_{2,d} & \epsilon_{3,d} \end{bmatrix}^T$ are, respectively, the desired task-space position and orientation, expressed as a quaternion. Therefore, the Cartesian task T_C can be defined as:

$$\begin{aligned} T_{C,p} =& \quad ({}^{b}\mathbf{J}_{d,p}, \dot{\mathbf{p}}_d + \mathbf{K}_p \mathbf{e}_p) \\ T_{C,o} =& \quad ({}^{b}\mathbf{J}_{d,o}, \boldsymbol{\omega}_d + \mathbf{K}_o \mathbf{e}_o) \end{aligned} \tag{4.12}$$

where $\mathbf{K}_p$ and $\mathbf{K}_o$ are positive definite weight matrices and $\boldsymbol{\xi}_d = \begin{bmatrix} \dot{\mathbf{p}}_d & \boldsymbol{\omega}_d \end{bmatrix}^T$ is the desired twist for the considered distal link.

Active Joint Mask

When executing a Cartesian task, usually all the joints of the involved kinematic chain are actively used. For certain tasks, it could be convenient to use just a sub-set of joints, leaving some of them in a locked state. For this purpose, given a set of joints to lock, the *Active Joint Mask* sets those joints' columns of the task Jacobian to zero, so that the related task becomes:

$$T_{mask} = (\mathbf{J}_{mask}, \dot{\mathbf{e}}^*) \tag{4.13}$$

where the i^{th} column of $\mathbf{J}_{mask}$ is defined as:

$$\mathbf{J}_{mask,c_i} = \begin{cases} \mathbf{J}_{c_i} & \text{if joint } i \text{ is active} \\ \mathbf{0} & \text{if joint } i \text{ is locked} \end{cases} \tag{4.14}$$

Constraints and Bounds

OpenSoT is able to manage several types of *constraints* and *bounds*:

Constraint	**Equation**
Unilateral constraint	$\mathbf{A}_c\dot{\mathbf{q}} \leq \mathbf{b}_u$
Bilateral constraint	$\mathbf{b}_l \leq \mathbf{A}_c\dot{\mathbf{q}} \leq \mathbf{b}_u$
Unilateral bounds	$\dot{\mathbf{q}} \leq \mathbf{b}_u$
Bilateral bounds	$\mathbf{b}_l \leq \dot{\mathbf{q}} \leq \mathbf{b}_u$

Table 4.1: Constraints and Bounds.

Since bilateral constraints and unilateral and bilateral bounds can be easily transformed in the form of unilateral constraints, it is possible to enforce them in the standard QP problem (4.8). For robotics applications, some fundamental constraints and bounds are imposed on:

- *Joint limits*: to avoid reaching physical joint limits;
- *Joint velocity limits*: to generate joint trajectories with bounded velocities;
- *CoM velocity limits*: to keep the cartesian velocity of the center of mass within imposed bounds.

Chapter 5

DARPA Robotics Challenge: the *Driving* Task

In this chapter, the implementation of the software module for the DRC driving task is presented[1]. The aim of this module is to control a humanoid robot, positioned inside a vehicle, that has to drive from a point A to a point B, while a remote operator sends high-level commands to it. The module is implemented as a GYM module, with a control loop thread frequency of 250 Hz.

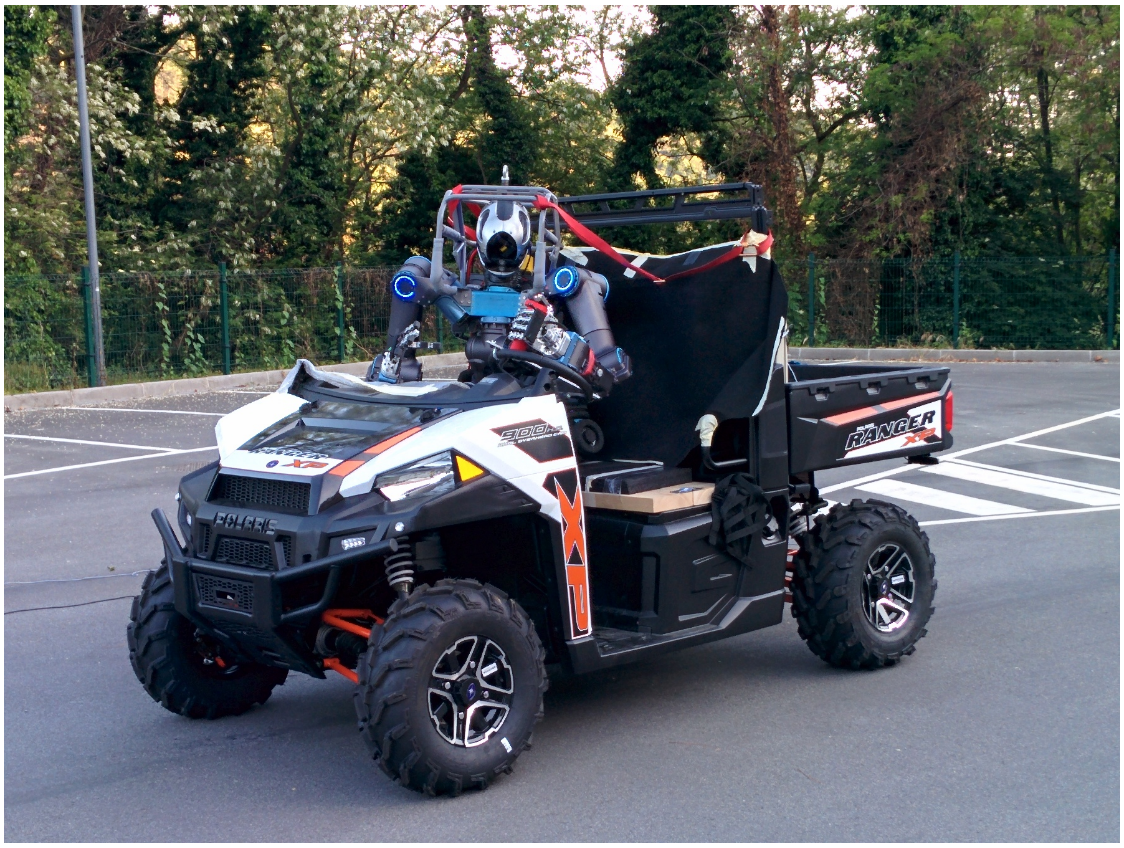


Figure 5.1: The Walk-Man robot in a driving configuration.

[1]The software module is open-source and available at `https://gitlab.robotology.eu/walkman-drc/drc_drive/tree/master`

5.1 Preliminary Robot Configuration

Before starting the driving module, as soon as the robot has been manually hauled inside the vehicle, the pilot issues a pre-configured pose to the robot, which acts at joint level through a direct position control. At this point, the robot is ready and the pilot can start the driving module.

5.2 Module Initialization

Through the *switch interface*, the operator starts the driving module on the robot, which is now ready for the initialization and to exchange data with the pilot interface. At this point, the Stack Of Tasks framework is initialized. In this module, the stack is composed of two cartesian tasks, as follows:

Task	**Distal Link**	**Base Link**
Accelerating	*walkman::LFoot*	*walkman::Waist*
Steering	*walkman::LSoftHand*	*walkman::Waist*

Table 5.1: Cartesian tasks composing the *Stack of Tasks*.

In addition, to complete the IK problem, the following constraints have to be satisfied:

Type	**Value**
Joint Limits	$q_i \in [q_i^{min}, q_i^{max}]$
Velocity Limits	0.7 m/s
CoM Velocity Constraint	0.3 m/s

Table 5.2: Constraints to be satisfied by the Inverse Kinematics solver.

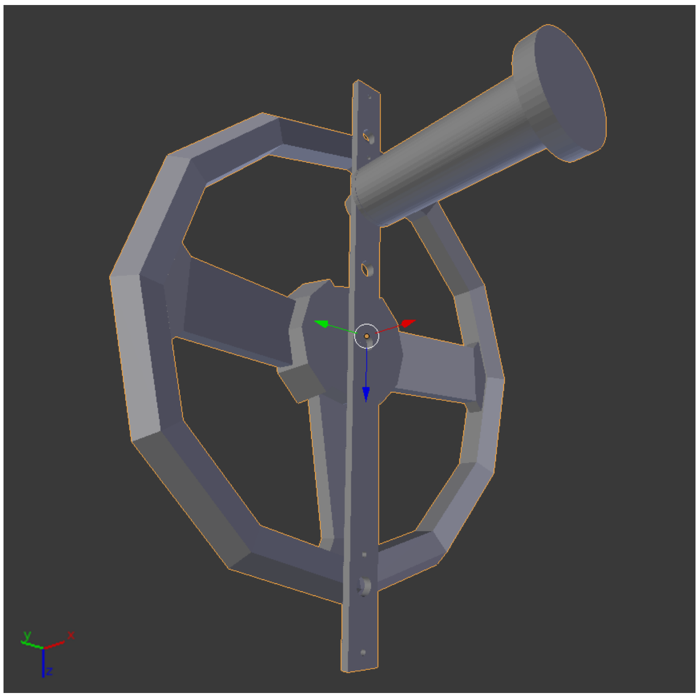
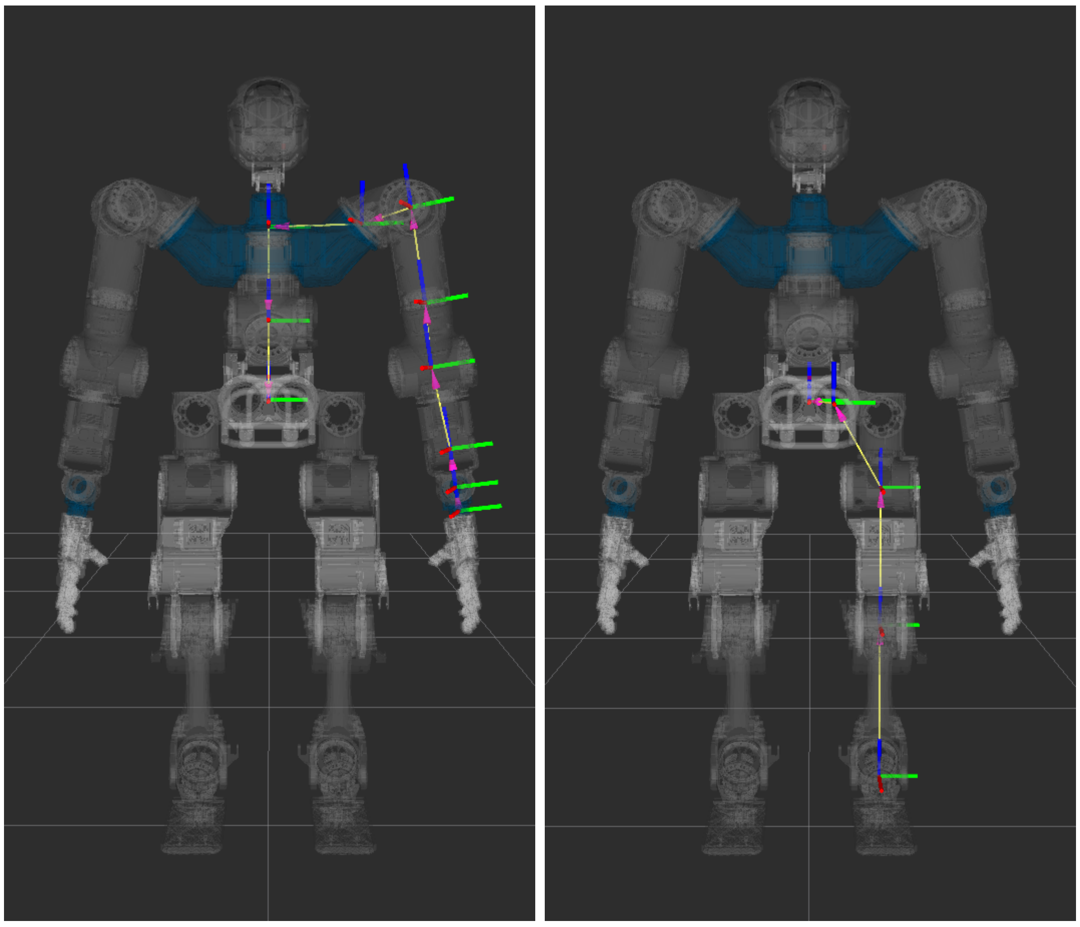


Figure 5.2: 3D model of the handle mounted on the steering wheel.

5.2.1 Steering Task

Regarding the *steering* task, it's clearly inconvenient for a humanoid robot to steer in a human way, with both hands on the steering wheel, because the task would become too much demanding in terms of complexity and degrees of freedom used. In addition, it would lead not only to the development and handling of a more complex control algorithm, but also a remarkable increase of power consumption. Taking inspiration from a technique used by truck drivers, a simpler and more convenient way of steering is that of using a handle attached to the steering wheel, allowing the steering task to be performed with a single hand. So, according to the robot's hand and steering wheel size, a handle has been designed, 3D printed, and provided with an independent rotation around its vertical axis (referred in Figure 5.2 as the x-axis). Furthermore, this solution allows also a simplification of the robot motion during the steering task, which can be performed only by using the left arm joints. To achieve this result, a mask of active joints (from the shoulder to the hand) is set on the task's Jacobian, which means setting to zero the matrix columns corresponding to the locked joints of the Waist-Hand kinematic chain (from the waist to the torso) (Figure 5.3). This operation allows smoother motions and reduces the possibility of collisions with vehicle parts due to the movement of the torso, leaving the robot still in its seat.



(a) *Steering* task. (b) *Accelerating* task.

Figure 5.3: Representation of the kinematic chains involved in the cartesian tasks of the driving module.

5.3 Control Loop Thread

The *run*() function of the control loop thread is the actual core of the module (Figure 5.4). At each step, the evolution of a Finite State Machine is handled and consequently the control law is applied on the robot. At the beginning, the state machine lays in an idle state, while the thread listens for external commands from the operator through the *command interface*. Those commands are composed of a header string and a payload of data, representing the command parameters that will be processed by a specific internal function. As soon as a valid command is received, the state machine commutes into the new state, which includes the computation of the trajectory planning for the requested behavior. Then, during the sensing phase, all the motor encoders are sensed on the robot and used as input of the control law. Finally, the new joint references are applied on the robot through a PID position control.

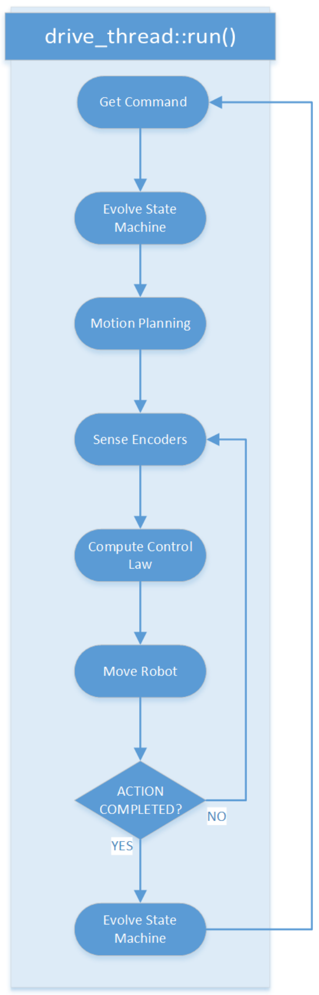


Figure 5.4: Flow diagram of the control loop thread.

5.4 Finite State Machine

The use of a discrete Finite State Machine allows to model a workflow in an event-driven manner (in this case, the event could be the receiving of a remote command or the completion of a planned motion). This architecture allows the robot to follow a precise flow of actions, avoiding incoherent robot behaviors and thus providing a safer control in teleoperation scenarios (Figure 5.5). In the following, all the states of the state machine are presented and explained in details.

5.4.1 State *idle*

This is the default state in which the state machine starts. On the pilot interface, the operator superimposes the steering wheel interactive marker over the corresponding

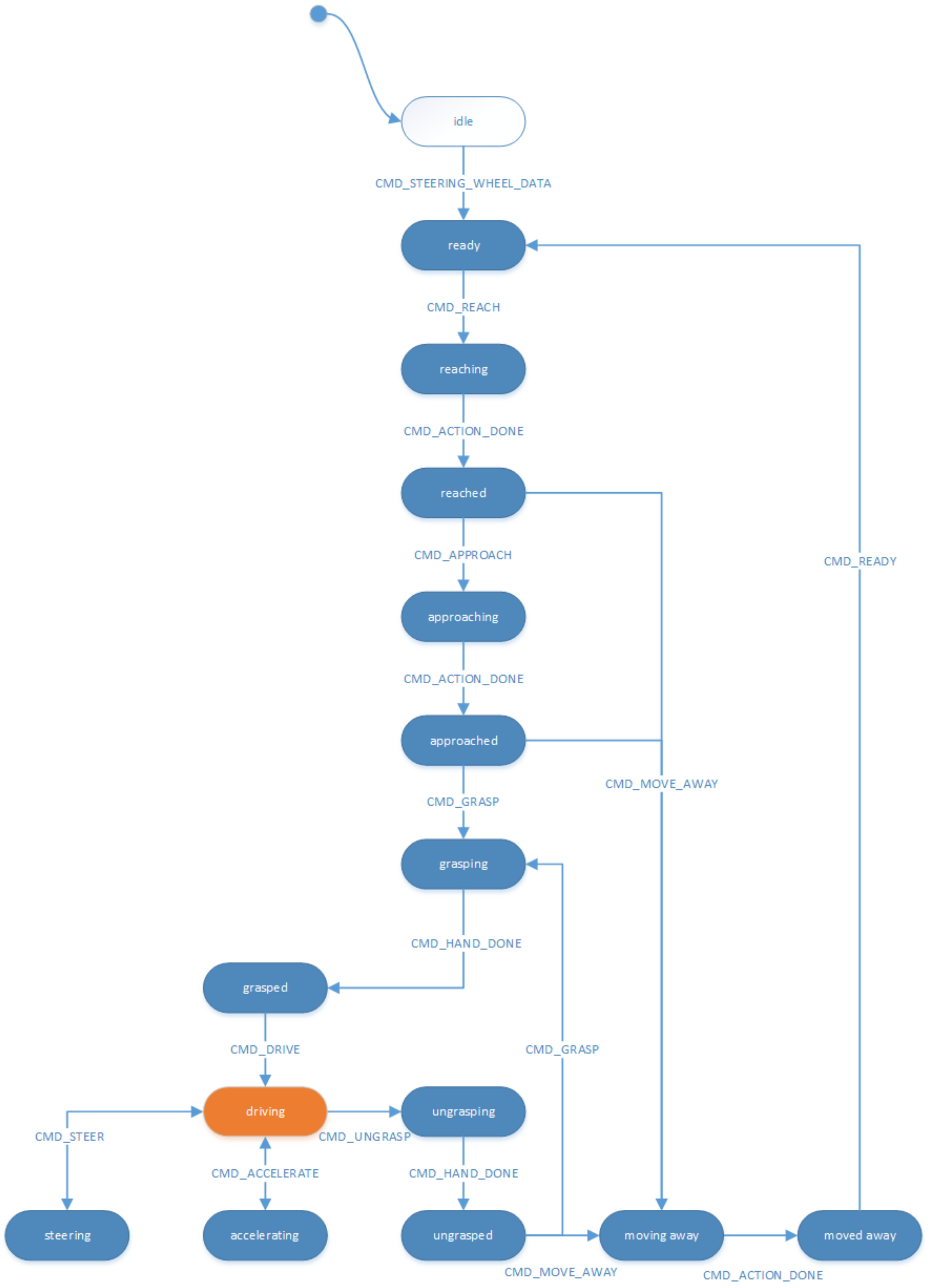


Figure 5.5: Workflow of the Finite State Machine.

object in the point cloud. Once done, the operator can issue the *set steering wheel* command, which also stores information about the steering wheel pose (position and orientation) with respect to a known robot frame, and it has the form of: 'set_steeringwheel *baselink* $x\ y\ z\ \phi\ \theta\ \psi$', where *baselink* is a string that specifies a robot frame used as reference frame, and the following parameters describe the steering wheel pose with respect to the base frame. Through some proper transformations, the steering wheel pose is processed in order to be referred to the same base-link frame of the cartesian tasks of the stack, for a coherent data handling. Then, since the position of the handle with respect to the steering wheel is known, it is possible to obtain its pose with respect to the *walkman::Waist* frame, using the post-multiplication rule:

$$^{\{waist\}}handle = {}^{\{waist\}}steeringwheel\,{}^{\{steeringwheel\}}handle$$

In addition, it is important to save the orientation of the steering wheel in its initial state, when the wheels are straight (referred as $^{\{waist\}}steeringwheel\_init$ in the following). By knowing this information, it is possible to fix the grasp position of the hand with respect to the steering handle, independently of the steering wheel angle. This generalization may turn out useful in the event of grasp loss of the steering wheel, followed by a new attempt of grasping it. In fact, in this case, the pilot would need to reposition the interactive marker according to the last steering wheel state and to re-issue the *set steering wheel* command, thus resulting in incoherent information passed to the successive states (i.e. *reaching* and *approaching*).

At this stage, all the information and the relative pose between the robot and the steering wheel are known.

5.4.2 States *reaching* and *approaching*

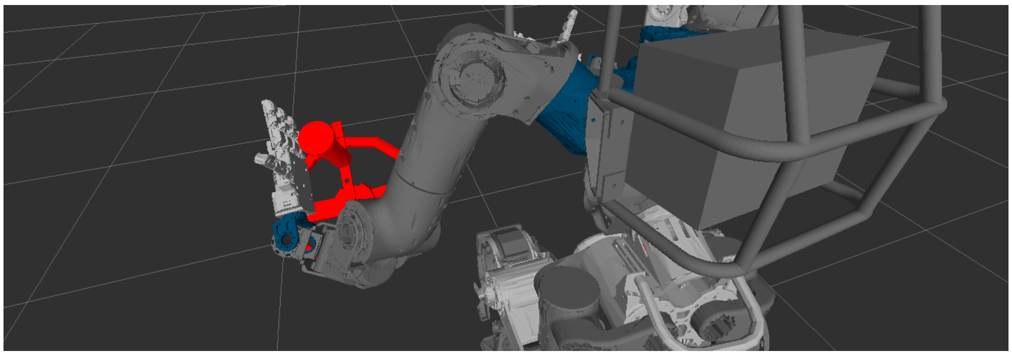
The robot's hand motion prior to the steering wheel grasp is separated into two phases: *reaching* and *approaching.* This solution provides a more robust handling of the motion and allows a backup plan in the event of trajectory issues or unwanted collisions. During the *reaching* phase, a linear trajectory is used to control the left hand of the robot and move it to the proximity of the handle mounted on the steering wheel, with the following target pose:

$$^{\{waist\}}LHand\_reach\_target = {}^{\{waist\}}handle\,{}^{\{handle\}}LHand\_reach\_target$$

Then, switching to the *approaching* phase, another linear motion brings the hand in contact with the handle, which is now ready to be grasped

$$^{\{waist\}}LHand\_approach\_target = {}^{\{waist\}}handle\,^{\{handle\}}LHand\_approach\_target$$

Regarding the axis orientation of both targets, as stated above, it is important to note that it is set equal to the one of $^{\{waist\}}steeringwheel\_init$, to get the hand well-aligned with respect to the handle.


(a) Robot state after *reaching* command.

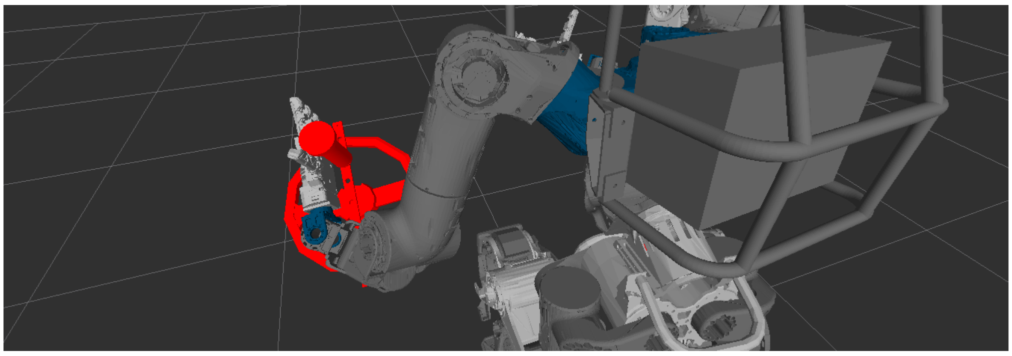

(b) Robot state after *approaching* command.

Figure 5.6: Details of *reaching* and *approaching* states in a simulated environment.

5.4.3 States *grasping* and *ungrasping*

In this state, the robot grasps the handle, controlling the single motor of the SoftHand (Figure 5.7). Since the hand is compliant by construction, it can adapt easily around the handle even in case of a small position/orientation error. Of course, an *ungrasping* procedure is also present. If the pilot believes that the grasp is not solid enough, he could decide to issue an *ungrasp* command and retry a new attempt of grasp. Otherwise, it is also possible to manually correct the hand position with the aim of the pilot interface. Once the handle is successfully grasped, the *get_rotation_radius*() routine computes the center of rotation of the circular trajectory used for the *steering* task, and consequently the radius of the rotation.

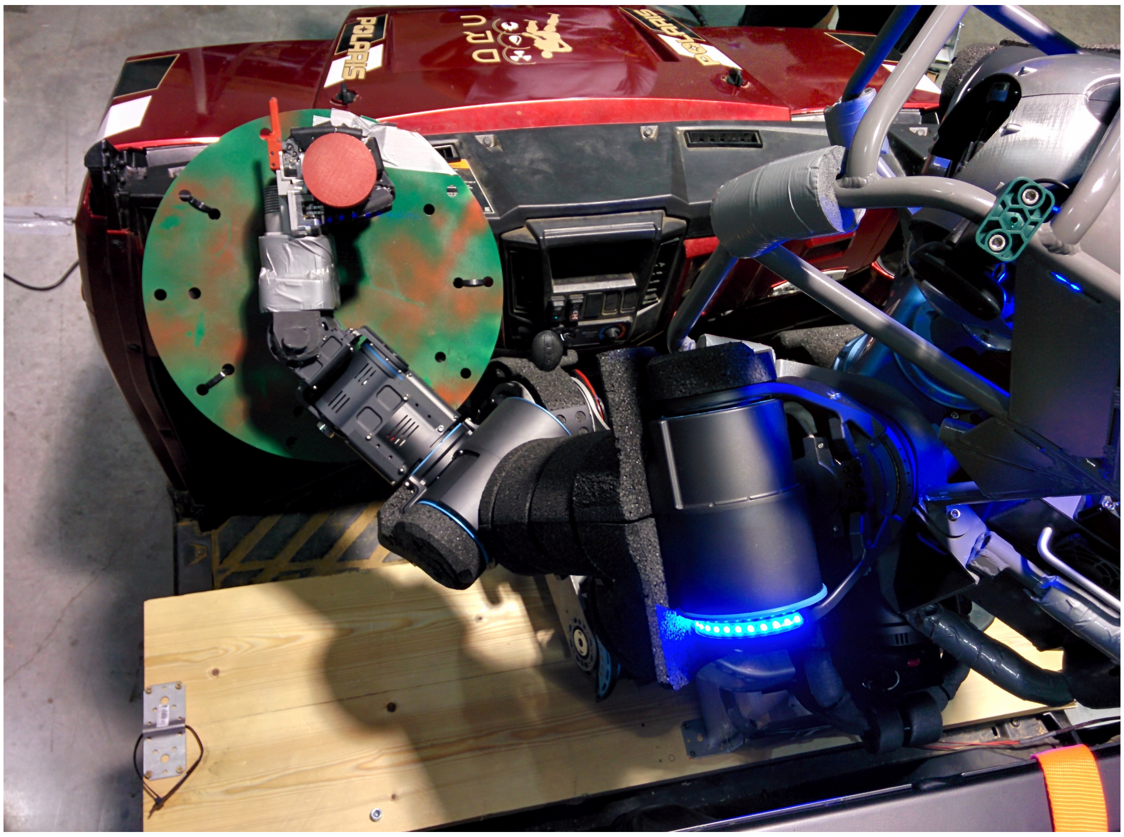


Figure 5.7: Details of the Walk-Man robot after grasping the steering handle.

Driving mode

At this point, the pilot can enter the *driving mode*, enabling the actual states that allow the robot to drive.

5.4.4 State *accelerating*

Every time the *accelerate* command is triggered, the robot pushes the throttle pedal by simply controlling the pitch of its left foot through a constant angular velocity trajectory. The pilot acts on two parameters that regulate the duration and the intensity of the throttle:

- *throttle time (s)*
- *throttle pitch (deg)*

The parameter *throttle time* specifies for how long the throttle pedal is pressed, while the parameter *throttle pitch* specifies the relative pitch angle of the foot. When *throttle time* has passed, the foot comes back to its home position. It is clear that for this particular task, the intermittent style of acceleration is more reliable and safe than a continuous acceleration (the human way).

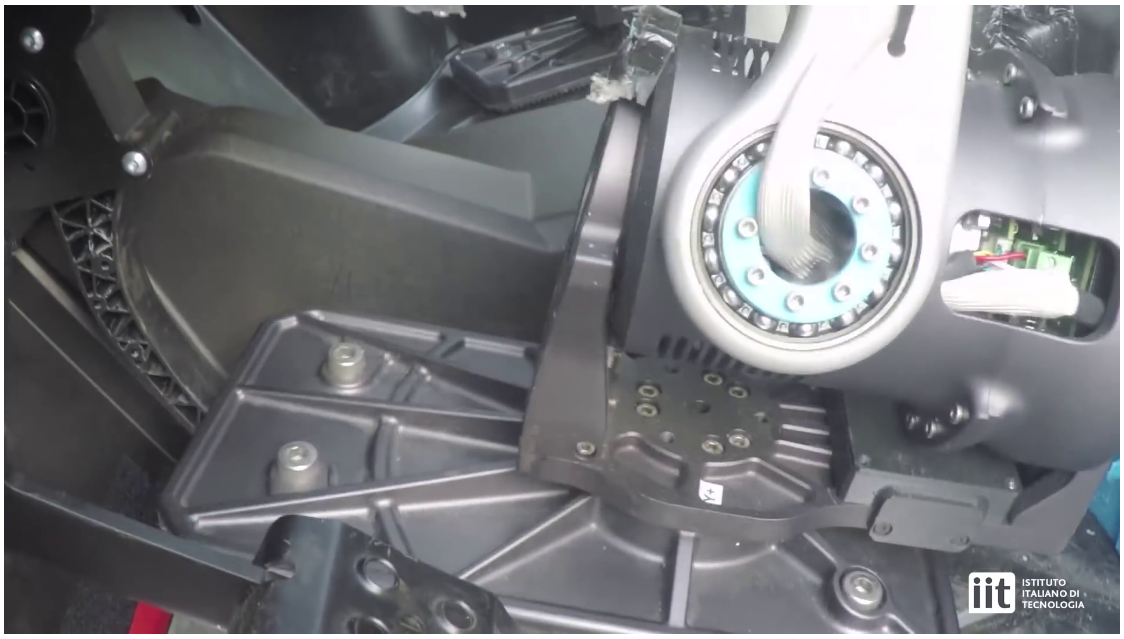


Figure 5.8: Details of Walk-Man's left foot while accelerating.

5.4.5 State *steering*

When the vehicle encounters an obstacle, the pilot can command the robot to bypass it by steering the vehicle. To accomplish this, a circular arc trajectory is issued on the robot's left hand, knowing the *rotation radius* and the *axis of rotation* from the *get_rotation_radius*() routine. The pilot acts on the following two parameters:

- *steering angle (deg)*
- *full circle time (s)*

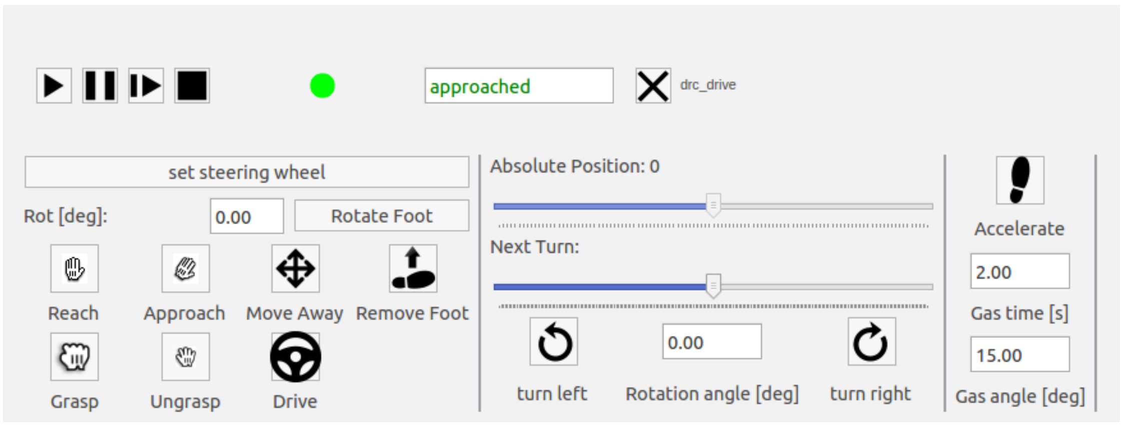
The parameter *steering angle* regulates the circular arc trajectory on the steering wheel, while the parameter *full circle time* defines the time within which the trajectory must be executed, hence its velocity. It is important to note that, during the whole trajectory, the orientation of the hand is kept fixed, following the axis of the $^{\{waist\}}steeringwheel\_init$ frame. This solution, even though adds a sort of stiffness on the wrist, it is feasible due to the free rotation of the steering handle.

Exiting *driving mode*

The only way to exit the *driving mode* is to issue the *ungrasp* command, followed by the *move hand away* command.

5.4.6 State *moving hand away*

To enforce robustness, the whole state machine is designed as a closed loop. For instance, hitting a bump in the road could cause an unwanted loss of grasp of the handle, making the robot current state incoherent within the module, which remains in *driving mode*. The pilot can easily recover from the afore-mentioned situation, by triggering the command *move hand away*, which sends back the robot's hand in a rest position, far from the steering wheel. So, without restarting the module, it is possible to repeat the commands in order to re-grasp the handle and resume driving.


approached

drc_drive

Rot [deg]: 0.00

Absolute Position: 0

Next Turn:

Rotation angle [deg]: 0.00

Gas time [s]: 2.00

Gas angle [deg]: 15.00

Figure 5.9: Details of the driving task dedicated widget on the Pilot Interface.

Chapter 6

Conclusions

The present work of thesis had the aim of extending the capabilities of the humanoid robot *Walk-Man*, in particular making it able to drive a vehicle. The presented implementation consisted in the development of a software module inserted in a teleoperation-oriented framework, where a pilot can remotely control the robot through a GUI.

6.1 DARPA Robotics Challenge Finals 2015

The goodness of the software module implemented on the robot platform was tested during the *DARPA Robotics Challenge Finals 2015* (Pomona, CA, 5-6 June 2015), a competition where participating teams and their robots confronted each others on a difficult course of eight tasks relevant to disaster response. The driving task was organized on a track that consisted of a 20m long and 6.5m wide course, with two side barriers as obstacles. The vehicle employed for the competition was a *Polaris Ranger XP900*, a compact off-road car with automatic transmission provided with a self-brake function. Each team had the possibility to add a custom equipment on the vehicle, but strictly without using any tool. Another rule regarded the robot center of mass, which had to fall inside the vehicle. The approach employed by the Walk-Man team for the driving task involved the use of two custom components:

- a camouflage-tinged plate mounted on the steering wheel through plastic strips, used as support for the steering handle (see Figure 5.7);
- a wedge, positioned under the gas pedal, to provide a physical speed limiter.

In addition, the driving module is designed in a way that the driving style would basically be a sequence of steering-accelerating tasks. In fact, the intermittent acceleration allows the operator to drive the vehicle with small steps, adjusting the direction of motion from time to time, according to the presence of obstacles or barriers.

During the DRC, the Walk-Man team managed to complete the driving task successfully in 8 minutes, without hitting any barrier.

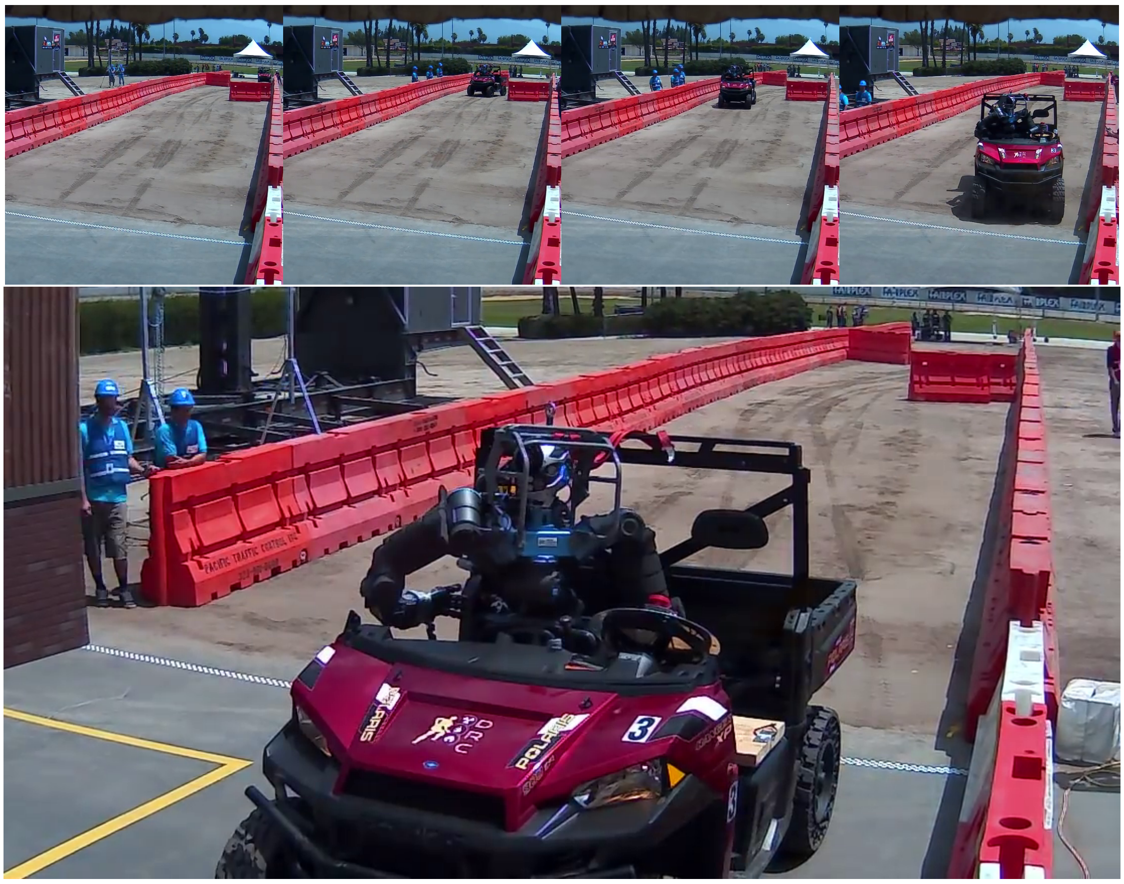


Figure 6.1: Time-lapse sequence of the DRC driving task performed by Walk-Man.

6.2 Beyond DRC: Lessons Learned and Future Works

The DRC event certainly showcased some examples of the state of the art in humanoid robotics, with also a significant experience gained on the field. Although several teams managed to successfully achieve all the eight tasks, it clearly resulted that the modern technology is not yet ready to allow an actual employment of these robots in a real disaster scenario. In fact, it is necessary to underline that the DRC environment was probably much more benign than Fukushima. For instance, radiation dose or simply water were not taken into account. Aside from this, another aspect that came out from the challenge is the fragilty of robot behaviors. Small variations in tools used for the manipulation tasks, or other unexpected changes in the enviroment, caused a lot of issues among almost all the teams. Not to mention several hardware or software failures, for which none of the teams were actually prepared.

However, it is important to state that, the robotics community is certainly going towards the right direction. The technology readiness level of the state of the art in robotics is actually not so far from a real operative state, but some critical aspects have to be improved. Regarding humanoid robots, the main aspect to enhance is the robot self-balance, taking into account different back up plans in case of falls. In addition, in order to improve the interaction with the environment and humans, better force control methods have to be reviewed. Also, a human-like skin sensing could provide robots superior compliance and interaction capabilities.

Bibliography

[1] Robin R Murphy, Satoshi Tadokoro, Daniele Nardi, Adam Jacoff, Paolo Fiorini, Howie Choset, and Aydan M Erkmen. "Search and rescue robotics". In: *Springer Handbook of Robotics*. Springer, 2008, pp. 1151–1173.

[2] Michael A Goodrich, Jacob W Crandall, and Emilia Barakova. "Teleoperation and beyond for assistive humanoid robots". In: *Reviews of Human factors and ergonomics* 9.1 (2013), pp. 175–226.

[3] *DARPA Robotics Challenge website*. `http://archive.darpa.mil/roboticschallenge/`.

[4] DRC-Teams. *What Happened at the DARPA Robotics Challenge?* `www.cs.cmu.edu/~cga/drc/events`. 2015.

[5] Università di Pisa and Istituto Italiano di Tecnologia (Genova). *WALK-MAN project website*. `https://www.walk-man.eu/`.

[6] Francesca Negrello, Manolo Garabini, Manuel G Catalano, Jörn Malzahn, Darwin G Caldwell, Antonio Bicchi, and Nikolaos G Tsagarakis. "A modular compliant actuator for emerging high performance and fall-resilient humanoids". In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE. 2015, pp. 414–420.

[7] Manuel G Catalano, Giorgio Grioli, Edoardo Farnioli, Alessandro Serio, Cristina Piazza, and Antonio Bicchi. "Adaptive synergies for the design and control of the Pisa/IIT SoftHand". In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782.

[8] Alessandro Settimi, Corrado Pavan, Valerio Varricchio, Mirko Ferrati, Enrico Mingo Hoffman, Alessio Rocchi, Kamilo Melo, Nikos G Tsagarakis, and Antonio Bicchi. "A modular approach for remote operation of humanoid robots in search and rescue scenarios". In: *International Workshop on Modelling and Simulation for Autonomous Systems*. Springer. 2014, pp. 192–205.

[9] Mirko Ferrati, Alessandro Settimi, Luca Muratore, Nikos G. Tsagarakis, Lorenzo Natale, and Lucia Pallottino. "The Walk-Man Robot Software Architecture". In: *Frontiers in Robotics and AI* (2016).

[10] RW Brockett. "Formal languages for motion description and map making". In: *Robotics* 41 (1990), pp. 181–191.

[11] Oussama Khatib. "A unified approach for motion and force control of robot manipulators: The operational space formulation". In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[12] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.

[13] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. "Yarp: Yet another robot platform". In: *International Journal of Advanced Robotic Systems* 3.1 (2006), p. 8.

[14] Luca Muratore, Mirko Ferrati, Enrico Mingo Hoffman, Alessio Rocchi, and Alessandro Settimi. *GYM repository*. `https://github.com/robotology-playground/GYM`.

[15] Arash Ajoudani, Jinoh Lee, Alessio Rocchi, Mirko Ferrati, Enrico Mingo Hoffman, Alessandro Settimi, Darwin G Caldwell, Antonio Bicchi, and Nikos G Tsagarakis. "A manipulation framework for compliant humanoid coman: Application to a valve turning task". In: *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE. 2014, pp. 664–670.

[16] Jinoh Lee, Arash Ajoudani, Enrico Mingo Hoffman, Alessio Rocchi, Alessandro Settimi, Mirko Ferrati, Antonio Bicchi, Nikolaos G Tsagarakis, and Darwin G Caldwell. "Upper-body impedance control with variable stiffness for a door opening task". In: *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE. 2014, pp. 713–719.

[17] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[18] Enrico Mingo Hoffman, Silvio Traversaro, Alessio Rocchi, Mirko Ferrati, Alessandro Settimi, Francesco Romano, Lorenzo Natale, Antonio Bicchi, Francesco Nori, and Nikos G Tsagarakis. “Yarp based plugins for gazebo simulator”. In: *International Workshop on Modelling and Simulation for Autonomous Systems*. Springer. 2014, pp. 333–346.

[19] Nicolas Mansard, Olivier Stasse, Paul Evrard, and Abderrahmane Kheddar. “A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks”. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on*. IEEE. 2009, pp. 1–6.

[20] Alessio Rocchi, Enrico Mingo Hoffman, Darwin G Caldwell, and Nikos G Tsagarakis. “Opensot: a whole-body control library for the compliant humanoid robot coman”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 6248–6253.