Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Laurea Magistrale in Ingegneria Biomedica

Vision-based high-level architecture for intraoperative supervision and behavioural control of a surgical robot

Relatore:
Prof. Giancarlo FERRIGNO

Correlatore:
Dott. Ing. Roberta PERRONE

Tesi di Laurea di:
Federico NESSI
Matricola 782877

Anno Accademico 2013-2014

Sommario

La robotica medica è un'area in continuo sviluppo. Se il punto di partenza per una simile applicazione era la possibilità di raggiungere livelli di accuratezza, velocità d'esecuzione e capacità di movimentare carichi pesanti impossibili per un essere umano, gli ultimi anni hanno visto la ricerca sempre più rivolta allo sviluppo di un assistente robotico più versatile. Una Sala Operatoria (SO) dotata di un sistema robotico è un ambiente critico in cui il robot chirurgico deve eseguire compiti ad alto rischio in uno spazio condiviso con l'equipe chirurgica. Un sistema in grado di gestire autonomamente una serie di situazioni d'emergenza può garantire all'equipe chirurgica una coabitazione più confortevole all'interno della SO, per esempio lasciando i medici liberi di aggirarsi attorno al robot anche durante il suo movimento. Quando l'equipe chirurgica è libera di muoversi attorno al robot, devono essere prese in considerazione emergenze legate alla perdita di informazioni relative alla posa intraoperatoria del robot (dovuta al fatto che i dati di *tracking* possono essere corrotti) così come la come le possibili collisioni del robot stesso con i chirurghi. I robot disponibili in commercio (ad esempio il NeuroMate o il Da Vinci) non possiedono un sistema di supervisione durante la fase intraoperatoria e di conseguenza si affidano completamente al chirurgo per il riconoscimento di eventuali emergenze.

In questo lavoro viene sviluppata un'architettura software ad alto livello in grado di supervisionare un sistema robotico semi-autonomo all'interno della sala operatoria. L'obiettivo è il raggiungimento di una più confortevole convivenza uomo-robot attraverso una robusta gestione di eventuali fallimenti dei sensori e

di situazioni di emergenza.

Poichè il movimento dell'equipe chirurgica può portare alla perdita di dati relativi al *tracking* intraoperatorio del robot (ad esempio dovute all'occlusione del campo di vista dei sensori), le emergenze relative al sistema di *tracking* devono essere rilevate e risolte autonomamente. Il sistema di tracking è stato sviluppato seguendo un approccio ridondante multi-sensore per la navigazione, in modo che qualora un sensore fallisca nel determinare la posa corrente del robot durante il movimento, il sistema possa fare affidamento su altri sensori di riserva. E' stata costruita una serie di Dynamical Reference Frame (corpi rigidi ai quali è associato un sistema di riferimento) in grado di essere visti da tutti i sensori considerati. I corrispettivi sensori sono poi stati calibrati tra loro e con il robot con un classico approccio *hand-eye*. E' stato disegnato uno schema ad albero (Fault-Tree Analysis) che identifichi i possibili fallimenti di un sensore. Infine, è stato implementato un set di componenti software in modo da gestire i dati in arrivo dai sensori e da restituire informazioni relative a quali sensori siano correntemente in grado di fornire la posa del robot.

Poichè il movimento dei medici attorno al robot può essere causa di collisioni, sono stati implementati la detezione e il *tracking* della figura umana all'interno dello spazio intraoperatorio, così come il rilevamento e la gestione di emergenze legate alle possibili collisioni. Il riconoscimento dell'equipe chirurgica è stato sviluppato usando delle camere RGB-D (che forniscono l'immagine RGB più un'informazione di profondità, *Depth*) in grado di riconoscere automaticamente la presenza di una persona (detta *utente*) all'interno del loro campo visivo. Dopo il riconoscimento, vengono costruiti una serie di sistemi di riferimento aventi l'origine nelle diverse articolazioni dell'utente. I dati relativi alla posa di questi sistemi di riferimento sono analizzati da un software appositamente scritto al fine di trovare corrispondenze tra diversi utenti con una metrica basata sulla distanza euclidea tra i sistemi di riferimento di ogni utente. Viene quindi calcolato un semplice inviluppo per

ogni utente, con sfere e cilindri costruiti attorno allo scheletro riconosciuto dalle camere.

Il comportamento del robot è continuamente aggiornato con le informazioni in arrivo dai moduli sopra descritti e sono eseguite reazioni alle emergenze (ad esempio, una frenata di emergenza) qualora uno dei moduli non sia in grado di eseguire correttamente il *tracking* o se viene rilevata una possibile collisione.

La supervisione di tutti i componenti coinvolti viene eseguita da tre diversi software (detti *Supervisor*, *Coordinator* e *Configurator*) che gestiscono rispettivamente: (1) i dati in arrivo da sensori e robot, (2) la conoscenza del compito da eseguire e (3) le azioni da compiere in seguito a particolari eventi. Il *Coordinator* è una macchina a stati gerarchica e rappresenta l'unico componente del sistema che ha conoscenza riguardo all'attuale procedura. Gli eventi necessari al *Coordinator* sono forniti dal *Supervisor* mentre le azioni da eseguire in seguito ad una particolare transizione di stato sono gestite dal *Configurator*.

Sono stati quindi eseguiti dei test all'interno di uno scenario semplificato (una procedura di approccio chirurgico, ad esempio durante la fase preliminare di una StereoElettroEncefalografia) al fine di valutare le performance dell'architettura. Il setup hardware comprende un KUKA LWR4+ come robot, un NDI Polaris Vicra e un NDI Optotrack Certus a comporre il sistema di tracking e due Microsoft Kinect come camere RGB-D. Il workflow adottato è il seguente: Dopo che il chirurgo ha dato inizio alla procedura, il sistema inizia il *tracking* del robot. Se il sensore con le migliori performance è in grado di seguire correttamente il movimento del robot, viene dato inizio al movimento. In caso di fallimento del sensore con le migliori performance, il secondo sensore viene connesso all'architettura e il movimento del robot è proseguito. Se nessun tracker è disponibile, viene eseguito uno stop di emergenza e il rilascio dei freni del robot deve essere confermato da un utente. In più, se viene rilevata una possibile collisione, il movimento del robot viene fermato e la ripresa delle operazioni deve essere confermata da un utente. In generale,

il robot si muove con due diverse velocità (*lenta* e *veloce*) a seconda della sua posizione nello spazio. La velocità *lenta* viene scelta se il robot si trova all'interno di un'area critica (ossia una sfera costruita attorno al target).

I risultati mostrano che è possibile effettuare *tracking* del robot anche intercambiando i sensori durante la procedura, con latenze di risposta nell'ordine dei 10 ms e con piccoli errori della posa rilevata dovute ai residui di calibrazione. L'algoritmo di riconoscimento della figura umana sviluppato è in grado di identificare con correttezza la presenza di un utente all'interno del campo operatorio in tutti i casi. Falsi positivi appaiono raramente e per brevi durate (minori di 1 s). La reazione ad una possibile collisione mostra una latenza nell'ordine di 1 ms prima dell'invio di un comando di stop al robot. Le performance in frenata sono fortemente dipendenti dalla velocità di movimento e mostrano uno spostamento del Tool Center Point rispetto alla posizione in cui è stata comandata la frenata che può raggiungere i 6 mm alla velocità di 10 cm/s. Alla velocità di 1 cm/s, lo spostamento in frenata è inferiore al millimetro.

Il sistema sviluppato, in condizioni di velocità di 7.5 cm/s per il movimento *veloce* e di 1.0 cm/s per il movimento *lento* è in grado di garantire reazioni con spostamento sub-millimetrico del Tool Center Point del robot per emergenze rilevate all'interno dell'area critica.

Sviluppi futuri possono riguardare l'implementazione di un inviluppo dipendente dall'operatore (ad esempio basandosi su mappe di profondità e non su sfere e cilindri) o di un comportamento più complesso del robot (ad esempio consentendo un controllo cooperato da parte dell'operatore quando sono soddisfatte particolari condizioni).

Parole chiave: robotica medica, sistemi di supervisione, controllori ad alto livello basati su sensori, gestione delle emergenze.

Abstract

Medical robotics is an application area in continous evolution. Started with the premise of reaching accuracy, speed and payload not possible to humans, in the last few years the resarch is aimed to provide a more versatile assistance system. Operating Rooms (OR) endowed with robotics systems are critical and dynamic environments in which the surgical robot must perform high-risk procedures, while cohabiting and interacting with members of the surgical equipe. A robotic system able to autonomously handle a set of defined emergencies that may occurr during the procedure can provide to the surgical equipe a more comfortable cohabitation inside the OR, for example leaving the surgeons free to move around the robot also during its movement. When the surgical equipe is free to move in the robot workspace, emergencies regarding the loss of data related to the robot pose in the intraoperative environment (e.g. due to something or someone that corrupts the tracking data) as well as possible collision with the surgeons must be taken into account. Commercially available robots (e.g. the NeuroMate or the Da Vinci System) does not provide an intraoperative supervision system and leave to the surgeon the management of eventual emergencies.

In this work we design a high-level software architecture for supervision and behavioural control of a semi-autonomous robot inside a crowded and shared environment like the OR. The final aim is to achieve a more comfortable human-robot cohabitation with surgeons free to move around the robot also during its movement through a robust management of sensor faults and emergency situations.

Because the surgeon movement can lead to the loss of intraoperative tracking

data (e.g. due to the occlusion of the Field Of View of the tracking sensors), emergencies related to the tracking system are detected and autonomously solved, when possible. A multi-sensor redundant approach for robot tracking is carried out so that if a sensor fails in determining the current 3D pose of the robot during its movement, the system can rely on other sensors as backup. A set of dedicated tools able to be seen from all the involved sensors have been built and sensors are calibrated with the robot and between each other using a classic *hand-eye* approach. A Fault-Tree Analysis is developed to analize all the possible causes that can lead to a sensor fault. A set of software modules handles the communication with the sensors and analizes incoming data to state at each moment if a particular sensor is correctly able to track the robot.

Because surgeons movement around the robot can cause collisions, human detection and tracking is performed inside the surgical workspace and emergencies related to collision are autonomously handled. Human detection is developed using a set of RGB-D cameras able to automatically detect and track people (called *users*) inside their field of view. After a user is detected, a set of Reference Frames having the origins in the user's joints is constructed. Incoming data from all cameras are analized in order to find correspondence between users using a metric based on euclidean distance between users Reference Frames. Once data of multiple Kinects are merged, a simple envelope (i.e. spheres and cylinders) is constructed around each user's skeleton.

The behaviour of the robot is updated with respect to information incoming from the above described modules and reactions to emergencies (e.g. the stop of the robot movement) are performed if the modules are no more able to provide tracking data or if a collision is detected.

Supervision of all the involved components is achieved using three different software supervisors (i.e. a *Supervisor*, a *Coordinator* and a *Configurator*) that handles respectevely (1) incoming data from both sensors and robot, (2) task

knowledge and (3) reactions to event and behavioural control. The *Coordinator* is actually a hierarchical state machine and represents the only task-aware component in the architecture. Driving events for the *Coordinator* are raised from the *Supervisor* and reaction to a transition is performed by the *Configurator*.

Tests are performed in a simplified scenario (i.e. a target approaching procedure, for example during the preliminary phase of a SEEG) in order to evaluate the architecture performance. The hardware setup includes a KUKA LWR4+ as the robot, a NDI Polaris Vicra and a NDI Optotrack Certus as the tracking system and a set of Microsoft Kinects as RGB-D cameras. The adopted workflow is the following: After the surgeon starts the procedure, the system begins to track the robot. If the tracker with the highest performance is able to correctly track the robot inside the OR, the movement is started. In case of fault of the best performant tracker, the second tracker is connected to the architecture and the robot movement is continued. If no tracker is available, the robot movement is stopped and in order to release the brakes, it is needed to ackowledge the event. Moreover, if a possible collision condition is detected (e.g. a user detected too close to the robot during movement) the stop of the robot movement is asserted and the release of the brakes needs to be aknowledged by a user. In general, the robot uses two different velocity of movement (i.e. *fast* and *slow*). The *slow* movement is performed when the robot is detected inside a critical area (i.e. a sphere around the provided target).

Results show that it is possible to track the robot also when switching between all the connected sensors with latencies of sensor swap in the order of 10 ms and with small oscillation in the detected pose due to calibration residuals. The developed human detection algorithm is able to correctly detect and track the surgeons inside the Operating Room. False positives in human detection rarely appear and for a limited amount of time (less than 1 s). Reaction to a possible collision shows a latency in the order of 1 ms before asserting the stop of the

robot motion. Braking performances are strongly dependant on the velocity of motion, with a displacement from the position where the stop command is asserted that can reach 6 mm at $10 cm/s$. At the velocity of 1.0 cm/s a sub-millimetric displacement in braking is shown.

The developed architecture, when moving at the two velocities of 7.5 cm/s for the *fast* movement and 1.0 cm/s for the *slow* movement, is able to provide sub-millimetric displacement or untracked movement of the robot Tool Center Point in reaction to emergencies detected inside the critical area.

Future works may be related to the implementation of a more user-specific envelope (e.g. based on point-clouds and not on spheres and cylinders) or a more complex behavioural control of the robot (e.g. allowing an *hands-on* control when particular conditions are fullfilled).

Keywords: medical robotics, supervision system, sensor-based high-level control, emergency management.

Contents

List of Figures XI
List of Tables XIV
List of Abbreviations XVII

1 Introduction **1**

1.1 Robot inside ORs: high-level control and supervision systems . . . 1
1.2 Aim of the work 3
1.3 Contents of the Thesis 4

2 State of the art **5**

2.1 Overview 5
2.2 Supervision systems in robotics 6
2.2.1 Robot supervision during surgical procedures 6
2.2.2 Examples of Supervision for surgical robots 7
2.3 High-level control design 9
2.3.1 Modularity in robotic architectures development 10
2.3.2 Examples of modular robotic architectures 11
2.4 Finite-state machines for robot behavioural control 14
2.4.1 Definition of finite-state machine 14
2.4.2 Examples of finite-state machine based robot behavioural controller 17

3 Materials and Methods **19**

3.1 Hardware description 20
3.1.1 Robot - KUKA LightWeightRobot 4+ 20
3.1.2 Tracker - NDI Optotrak Certus® 22
3.1.3 Tracker - NDI Polaris Vicra 23
3.1.4 Tracker - Microsoft Kinect 25
3.2 Software tools 26
3.2.1 ROS and OROCOS 26
3.2.2 rFSM 30
3.3 Multi-sensor robot tracking 32
3.3.1 Tool tracking 32
3.3.2 Calibration 33
3.3.3 Sensor Fault Analysis 36
3.3.4 Multi-sensor control architecture 39
3.4 Multi-kinect people tracking 45
3.4.1 People detection 45
3.4.2 People tracking 46
3.4.3 Calibration 48
3.4.4 Multi-kinect control architecture 49
3.5 Robot Behavioural control 54
3.5.1 Scenario 54
3.5.2 Finite-state machine description 55
3.5.3 Behavioural control architecture 57
3.6 Application 63
3.6.1 Complete architecture 63
3.6.2 Graphical User Interface 66
3.6.3 Performance tests 68

4 Results **75**

4.1 Overview 75

4.2 Multi-sensor robot tracking 75
4.3 Multi-Kinect people detection 79
4.4 Robot behavioural control 81

5 Discussions and conclusions 85

Bibliography 90

List of Figures

2.1 Op:Sense Framework 8
2.2 Triskar2 robot. Control architecture 11
2.3 Intuitive® Telesurgery System. Control architecture 12
2.4 NOTES robot semi-autonomous movement architecture. 13
2.5 Turnstile finite-state machine 15
2.6 Tape-recorder finite-state machine 16
2.7 Workflow for a laser bone ablation procedure 17
2.8 Scrub Nurse Robot. Statecharts representation 18

3.1 KUKA LWR 4+ and Controllers 21
3.2 NDI Optotrak Certus® Field of View 23
3.3 NDI Polaris® Vicra Field of View 24
3.4 Microsoft Kinect® Sensor 25
3.5 ROS basics. Nodes and topics representation 27
3.6 OROCOS. Schematic view of a Component 28
3.7 OROCOS. ExecutionEngine finite-state machine 29
3.8 rFSM Transitions Performances 31
3.9 Developed Optical Tools 33
3.10 Calibration Tools and involved transformations 34
3.11 Sensor Fault-Tree Analysis 38
3.12 Multi-sensor tracking architecture 40
3.13 Sensor Manager. Data Analysis Flowchart 44

3.14 User's joint reference frames 47
3.15 Calibration Tools and involved transformations 48
3.16 Multi-sensor people tracking architecture 50
3.17 Workspace People Detector Flowchart 53
3.18 System Behaviour Finite-State Machine 56
3.19 Behavioural Control Architecture 58
3.20 Developed rFSM finite-state machine for Robot Behavioural Control 60
3.21 Complete Control Architecture 65
3.22 Graphical User Interface 67

4.1 Results. Sensor fault reaction latencies 76
4.2 Results. Example of sensor swap during robot movement 77
4.3 Results. Repeated sensor fault reaction latencies 78
4.4 Results. Calibration error for a couple of Kinect 80
4.5 Results. Collision reaction latencies 80
4.6 Results. Example of the trajectory commanded to the robot . . . 82
4.7 Results. Example of emergency `STOP` command and TCP displacement 83
4.8 Results. TCP Displacement after `STOP` assertion 84

List of Tables

3.1 KUKA LWR 4+ Specification 21

3.2 NDI Optotrak Certus® Specifications 22

3.3 NDI Polaris®$^{\mathrm{Vicra}}$ Specifications 24

3.4 Robot-Tracker calibration specifications and results 35

4.1 Results. Multi-Kinect people detection performances 79

4.2 Results. Multi-Kinect people detection accuracy, sensitivity and specificity 79

List of Abbreviations

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
CTO	Calibration Tool Optical
DRF	Dynamic Reference Frame
FOV	Field Of View
FRI	Fast Research Interface
FTA	Fault-Tree Analysis
LWR	Light Weight Robot
OR	Operating Room
ORoCoS	Open Robot Control Software
RF	Reference Frame
ROS	Robot Operating System
RTO	Reference Tool Optical
TCP	Tool Center Point
${}^{A}\mathbf{T}_{B}$	Pose of A in the reference frame of B
KRC	KUKA Robot Controller

1 Introduction

1.1 Robot inside ORs: high-level control and supervision systems
1.2 Aim of the work
1.3 Contents of the Thesis

1.1 Robot inside ORs: high-level control and supervision systems

Medical robotics is an almost 30 years old application in which approaches and objectives are rapidly evolving [1, 2]. Operating Rooms (OR) endowed with robotics systems are critical and dynamic environments in which the surgical robot must perform high-risk procedures, while cohabiting and interacting with members of the surgical equipe. Because of this, current state of the art in surgical robotics shows that, despite significant improvements in efficiency and workflow, set up time in robot assisted surgery makes procedures lengthier than their non-robotic counterpart [3]. In particular, human-machine interaction is a critical point and an ideal protocol for safety maintainance is still not estabilished. Thus, current robotics research is aimed to provide more versatile assistance systems. A robotic system able to autonomously handle a set of defined emergencies that may occurr during the procedure can provide the surgical equipe with a more comfortable cohabitation inside the OR, for example leaving the surgeons free to move around

the robot also during its movement.

Autonomy of a robotic system is a concept strictly related to the ability of the perception of the surrounding enviroment and the corresponding adaptation of the robot behaviour to the new parameters that comes up [4]. Inside the OR, the state of the robot must be continously checked and updated with respect to the current step of the procedure (i.e. the workflow), the surgical equipe position inside the workspace and the possible emergencies stated above. If the surgical equipe is free to move in the robot workspace also during its movement, emergencies regarding the loss of data related to the robot pose in the intraoperative environment (e.g. due to something or someone that corrupts the tracking data) as well as possible collision with the surgeons must be taken into account. Thus, high-level software controller able to supervise the overall system shall provide an intraoperative tracking system that can robustly provide the surgical robot position even in a crowded environment such as the OR. Moreover, information about the workspace status (i.e. detection and tracking of the surgical equipe) shall be provided in order to transparently update the robot behaviour with respect to situations happening around it. The emergencies stated above must be detected and the behaviour of the surgical robot shall be modified in reaction to them.

Commercially available surgical robots such as the NeuroMate (Renishaw[1], United Kingdom) or the da Vinci System (Intuitive Surgical[2], Sunnywale, California) does not provide an intraoperative supervision system and completely rely on the surgeon for acknowledgment of emergency situation and consequent actions (for example, the NeuroMate system ensures check for collision with the stereotactic frame only during the preoperative phase).

[1] `http://www.renishaw.com/`
[2] `http://www.intuitivesurgical.com/`

1.2 Aim of the work

In this work we design a high-level software architecture for supervision and behavioural control of a semi-autonomous robot inside a crowded and shared environment like the OR. The final aim is to achieve a more comfortable human-robot cohabitation with surgeons free to move around the robot also during its movement. Because the surgeon movement can lead to the loss of intraoperative tracking data (e.g. due to a surgeon covering the Field Of View of the tracking system), emergencies related to the tracking system are detected and autonomously solved, when possible. Moreover, because surgeons moving around the robot can cause collisions, human detection and tracking is performed inside the surgical workspace and emergencies related to collision are autonomously handled. The complete high-level architecture is developed considering modularity, hardware independancy and hard software re-usability as constraints.

Intraoperative tracking of the robot is guaranteed with a multi-sensor approach in order to provide robustness to a single sensor fault. Strategies for multi-sensor tracking are developed and used to write a software module that manages the incoming data and provide the overall system with the needed tracking information.

The OR is detected using a set of RGB-D (i.e. RGB and Depth information) cameras in order to perform human detection and tracking, so that collision between the robot movement and the surgical equipe are avoided. Multi-Kinect people detection is developed and strategies to write a software module that recognize the surgical equipe and find correspondences between data incoming from the different sensors are used.

The behavioural control of the robot is achieved using a stand-alone finite-state machine that completely describes the task to be performed and controls the system status at every moment. Finally, we applied the developed control to a simple surgical scenario (i.e. the preliminary phase of a needle insertion, for example during a SEEG procedure) and performed tests to evaluate the performances.

1.3 Contents of the Thesis

The outline of the thesis is the following:

- In *Chapter 2* the state of the art in supervision systems for surgical robotics, togheter with the basic theory and examples from literature regarding the strategies we will use to design the architecture aim of this work are presented;

- In *Chapter 3* hardware setup and software tools are described. The complete strategies developed to perform multi-sensor tracking and multi-Kinect human detection are described togheter with the software modules that runtime handle the task. Emergencies are described and the behavioural control of the robot is presented with the case study application and the performed tests;

- In *Chapter 4* results of the tests performed in the described scenario are reported;

- In *Chapter 5* discussions, conclusions and future works are presented.

2 | State of the art

2.1 Overview

2.2 Supervision systems in robotics

2.2.1 Robot supervision during surgical procedures

2.2.2 Examples of Supervision for surgical robots

2.3 High-level control design

2.3.1 Modularity in robotic architectures development

2.3.2 Examples of modular robotic architectures

2.4 Finite-state machines for robot behavioural control

2.4.1 Definition of finite-state machine

2.4.2 Examples of finite-state machine based robot behavioural controller

2.1 Overview

In this chapter a description of the state of the art in supervision systems for surgical robotics is presented. Then, methodologies used to achieve the intraoperative supervision, thus software design strategies able to enhance re-usability and hardware independancy (i.e. the possibility to be simply used in a great variety of contexts) and high-level behavioural control in robotics with finite-state machines, are described.

2.2 Supervision systems in robotics

Robotic systems are complex hardware and software architectures usually developed by coupling different sensors (e.g. position sensors, force sensors, cameras, etc.) and actuators (i.e. the robot manipulator itself). Thus, the processing of a huge amount of data incoming from different sources has to be taken in account. These data are needed in order to retrieve information about the actual state of the system, thus the robot itself (*internal state*) and the surrounding environment (*external state*) [4].

A *supervisions system* is a high-level controller that manages the data incoming from all the different sources in order to react to the current situation of both the external and the internal state changing the behaviour of the involved agents. *Supervisors* can be designed using a wide range of approaches and in order to provide a great variety of tasks such as, for example, a system that is human-aware [5] or includes cognitive process to handle gesture-commands from a user [6].

2.2.1 Robot supervision during surgical procedures

A *Supervisor* is a primary need when deploying robotic architectures inside sensible *non-static* enviroments that involve contemporary presence of both robots and humans.

An OR equipped with a robotic system is an environment in which humans (i.e. the surgical equipe) and robots cohabits when executing a high precision task in a critical situation. Data incoming from different sources (e.g. navigation systems, workspace detection, etc.) needs to be merged in order to obtain information about the current status of all the involved systems. Safety and continuity of the information dataflow is crucial, as well as the ability to robustly react to emergency situations that may arise.

Nevertheless, most of the commercially available robotic systems (both au-

tonomous or teleoperation-based), such as the CASPAR® (URS Ortho GmbH & Co. KG, Germany) system used for total hip replacement or the ROBODOC®[1] system (Mission Falls Court, Fremont) used for knee replacement, does not provide a supervision system [7].

2.2.2 Examples of Supervision for surgical robots

Next we provide some examples of general purpose Supervision systems for surgical robotics architecture. A wide number of *Supervisor*s has been developed outside the surgical usage but they do not enter the target of this work and thus they are not described here in a detailed way.

Op:Sense Framework. The Op:Sense system is a *Supervision system* for intraoperative control of a telemanipulated surgical robotic setup developed at the Institute for Process Control and Robotics of Kalsruhe Institute of Technology[2] (KIT) and presented in [8]. The setup shows two LWR4+ robots (KUKA, Germany) that act like slave robots and a RX90 robot (Stäubli) that position an endoscopic camera on the operating field. PMD and RGB-D cameras are used to perform navigation and workspace detection and data are retrieved using CORBA[3]. All cameras are registered to each other, allowing computation of a live three dimensional voxel-based reconstruction of the scene which can be used for several purposes (see Fig. 3). By dynamically updating the robots' trajectory based on the obstacles in the scene, it's possible to avoid collisions between robots and humans. Controlling the devices is reached inside Simulink environment (MathWorks) and a feedback to the user is provided using a Graphical User Interface that shows both the tracked environment and the status of the system. A YAWL (Yet Another Workflow Language[4]) description of the procedure workflow is used to retrieve knowledge about the current step and to update the system. A setup

[1] http://www.robodoc.com/
[2] http://www.kit.edu/
[3] http://www.corba.org/
[4] http://www.yawlfoundation.org/

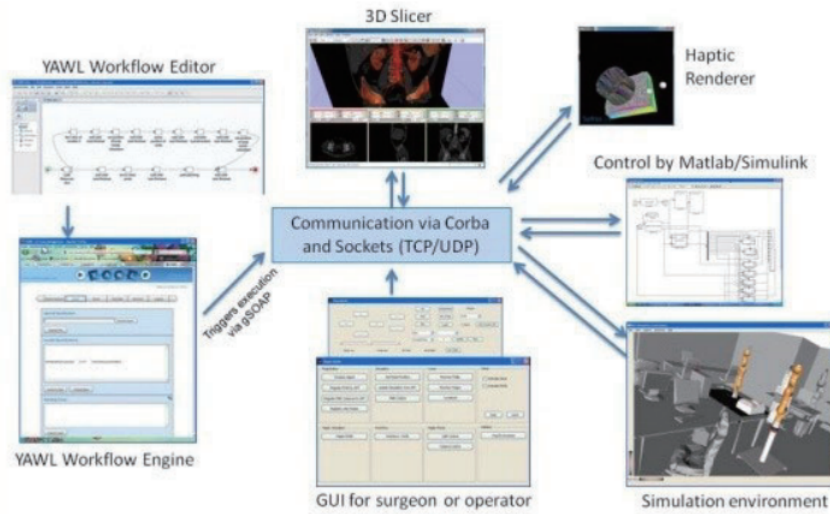


Figure 2.1: Op:Sense Framework. *Robot control is achieved inside Matlab environment, that provides both simulation and actual control. A graphical user interface is developed to handle user's inputs and to provide a visual feedback to the operator. A Workflow manager controls the different steps required in a defined procedure. Image taken from [8]*

planner was developed and is used to define the initial poses for the robots. The complete setup of the system is shown in Figure 2.1.

ACTIVE System Behaviour Supervisor. The *System Behaviour Supervisor* application is a *Supervision system* developed at NearLab (Politecnico di Milano) within the ACTIVE Project[5] (FP7-ICT-2009-6-270460) in order to switch the behaviour of a robotic setup for neurosurgical intervention with an events-based approach. The scenario shows two KUKA LWR4+ robots that act in different control modality (i.e. autonomous, cooperated and teleoperated). The system is presented in [9]. The behaviour of each agent in the scenario is controlled by a dedicated finite-state machine that is updated by a high-level Workflow manager that has knowledge about the procedure to be accomplished and handles the switch from one step to another. The system works over CORBA enviroment and controls the robots using ROS and OROCOS framework.

[5]`http://www.active-fp7.eu/`

2.3 High-level control design

Thinking of human beings, the most used method to solve complex tasks is to break it up into more simple and performable ones. Accordingly, the design of a software that needs to interact with many agents of different nature in order to perform some tasks can be accomplished with a similar, thus *modular*, approach.

As stated by C. Baldwin and K. Clark in [10]:

> « *A module is a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units.* »

A module is a single cell of a larger system that works structurally in an independent way, but conveys to produce a major task when put togheter with other cells. So, a *modular system* is a software built up starting from simple software *modules*. Thinking this way, a definition of *architecture* is provided by P. Clements, [11] as:

> « *The set of structures needed to reason about the software system, which comprise the software elements, the relations between them, and the properties of both elements and relations.* »

In a certain way, an *architecture* is an ensemble of the *modules* needed to achieve a goal (e.g. a task to perform) and the relations between those modules, such as connections and exchanged data. *Modular programming* is widely known as the top-down approach to write software architectures using modules as starting units. Its main advantages are stated ([12]) as:

- **clarity**: understanding the functionality of a single module of code is often easier than understanding a complex task performing single program.

- **re-usability** and **flexibility**: if different applications share even only one common action (for example, *to compute the inverse of a matrix*) the developed module can be simply inserted into both applications without the need to modify the code (*re-usability*). Therefore, if there is need to change part of the application, it is often necessary to substitute only a module with another one (*flexibility*).

On the other hand, the programmer has to take care of the *deployment* of the architecture (i.e. loading the needed modules, connectig them and executing them) instead of only launching a single executable file.

2.3.1 Modularity in robotic architectures development

Because of its advantages, the modular approach is particularly suitable to design software architectures to be used in robots control. In fact, the velocity of production of new robot controllers strongly depends on the amount of already available software that can be reused from former projects [13].

The ability to develop *atomic* software modules for basic robot control (e.g. low-level modules that compute forward and inverse kinematics, manage redundancy, perform admittance control, etc.) as well as the possibility to provide high-level modules able to supervise other components, coordinate their behaviour and interface with an external *workflow* descriptor is then crucial. In fact, this allows to dispose of simple software modules handling simple tasks and able to be connected to produce more complex behaviours. As shown in the following examples, connection between such kind of simple modules leads to design an architecture able to control robotics system even in complex task, while maintaining flexibility and hard software reusability.

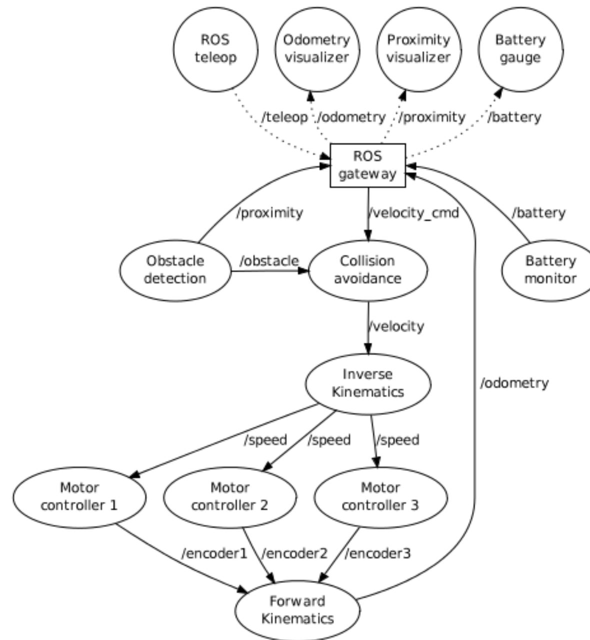


Figure 2.2: Triskar2 robot control architecture. *Modules are represented by circles and ellipses, while arrows represent connections between modules and exchanged data. This architecture implements a closed loop control, also providing collision avoidance. Every atomic task is performed by a single module. Image taken from [14].*

2.3.2 Examples of modular robotic architectures

Here we provide examples of modular softwares developed in literature to control robotic systems. Since our target is the development of an architecture to be used in surgical context, we paid particular attention to those architectures which are linked to surgical applications.

Triskar2 control architecture for R2P framework test. Rapid Robot Prototyping (R2P) is a framework developed by AIRLab (Department of Electronics, Information and Bioengineering, DEIB, Politecnico di Milano[6]), in order to achieve rapid robot prototyping through modularity. In order to test the developed framework, in [14] a software (and hardware) architecture to control Triskar2 robot is presented. Triskar2 is an omnidirectional robot that moves on wheels and it represents the only *non-surgical* application we will focus on.

[6] `http://www.deib.polimi.it/`

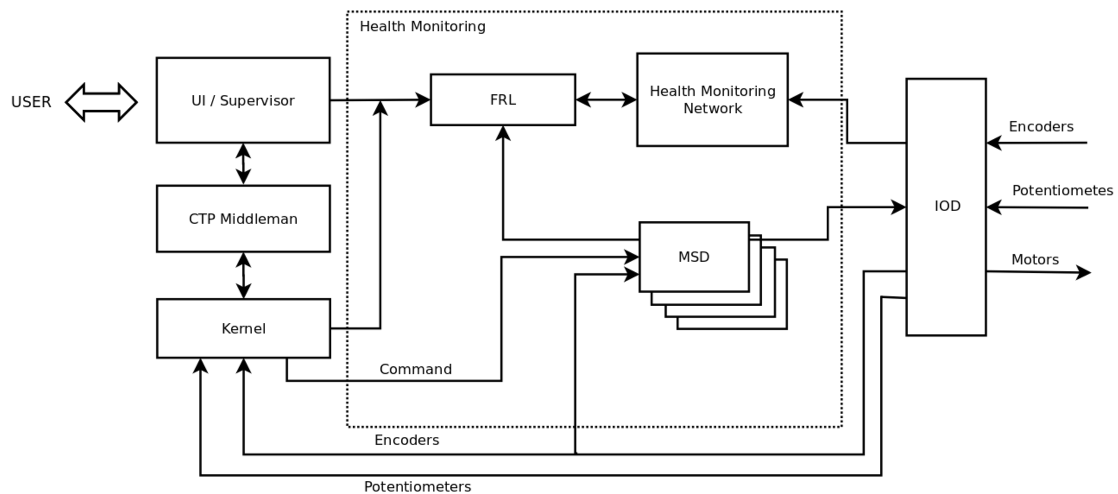


Figure 2.3: Intuitive® Telesurgery System. Control Architecture. *The User Interface is provided by* UI / Supervisor *module, that also handles the inputs.* Middleman *takes care of mathematical calculation and forwards the results to the* Kernel *module. This finally provides the control signal to* Multiple Servo Driver *(MSD) modules that control the drives. A high-level control (*Health Monitoring *module) handles exceptions and errors. Image re-elaborated from [15].*

The presented architecture, shown in Figure 2.2, aimed to teleoperated control, is built using C++ modules and deployed in a framework able to provide communication between modules using a publisher-subscriber protocol (ROS, *Robotic Operative System*[7]). Robot target command is issued by *ROS Teleop* module, that generates a velocity command checked by *Collision Avoidance* module. Information required by the *Collision Avoidance* module are provided by the *Obstacles detection* module. Once the velocity command is checked, it is sent to the *Inverse Kinematics* module, which provides speed command to the robot actuators. In order to obtain a closed-loop control, the actual position of the drivers is read by encoders then sent to the *Forward Kinematics* module and forwarded back to the Teleoperation Controller.

Intuitive® Telesurgery System. Intuitive Surgical® Inc.[8] has developed the most successful robotic system for surgery: the DaVinci® Robot. The software architecture that provides teleoperation to the DaVinci Robot is developed fol-

[7] http://www.ros.org/
[8] http://www.intuitivesurgical.com/

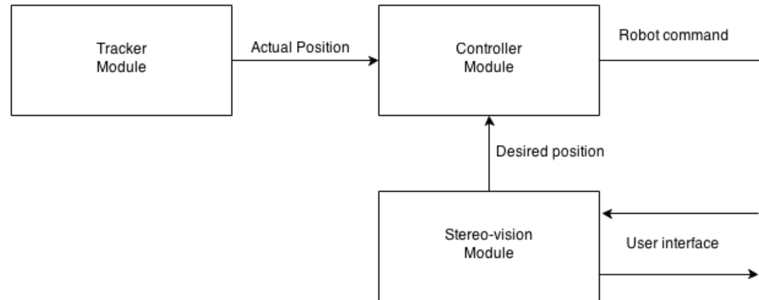


Figure 2.4: *NOTES robot semi-autonomous movement architecture. The* Tracker module *performs navigation and compute the position of the robot joints using a marker-based approach. The user input is actually a target on the image returned from the* Stereo-vision module *that is computed as a* go to *command for the robot. Control of the movement is performed with a basic PID contained into the* Controller Module*. Image elaborated from [16].*

lowing a modular approach, thus represents a relevant example for surgical robotic control architecture. As reported in [15], the control architecture is composed by (Figure 2.3) a *User Interface / Supervisor* module that represents the surgeon interface with the system and contains the handling of user's inputs; a *Middleman* component that take care of principal mathematical calculations to support and enable smooth transitions in servo mode; a *Kernel* module which closes the loop. Each *Multiple Servo Driver* (MSD) module controls the servo drives and the higher-level controller *Health monitoring* module handles errors and exceptions.

NOTES robot for a semi-autonomous surgical task control. Natural Orifice Translumenal Endoscopic Surgery (NOTES) robots are developed to perform *in vivo* minimally invasive surgical endoscopic procedures. A study reported in [16] developed a modular architecture that allows this kind of robots to perform simple *semi-autonomous* task such as the approach to a specific target defined by the surgeon on the stereo-vision monitor. A functionality overview shows that, after the user has provided the system with a desired position to be reached, the target approaching is performed measuring each joint position with a *Tracker* module that handles data incoming from an optical sensor. A representation of the architecture is reported in Figure 2.4.

2.4 Finite-state machines for robot behavioural control

Once the designer has provided a surrounding software able to process the sensor readings, manage communications with other agents, interface the hardware and control the actuators, there is the need to provide a behavioural controller (*task descriptor*) that manages the *decision making process* and tells the system how to react to different situations.

There are different ways to approach this task, but their description is far from the aim of this work (introduction to [17] can act like a simple review). Here we only want to state that *extensibility* and *reusability* of behaviours in different context are keys issues [17]: the possibility to integrate new behaviour or to re-use developed behaviours in different context and applications can be time-safer and guarantee to easily modify a system behaviour changing only the task descriptor.

2.4.1 Definition of finite-state machine

Here we describe finite-state machines as a modular and extensible approach to system behavioural control. The following dissertation is mainly based on [18].

Formally, a finite-state machine is a mathematical model used to design both computer programs and sequential logic circuits. It represents an abstract machine that can only have a finite number of *states*, with only one active (*current*) state at a time and a defined number of *transitions* that describes what condition(s) triggers the change from one state to another.

Next we provide definitions of *state* and *transition*:

- **State.** A description of the status of a system that is waiting to execute a transition.

- **Transition.** A set of actions to be executed when a condition is fulfilled or when an event is received.

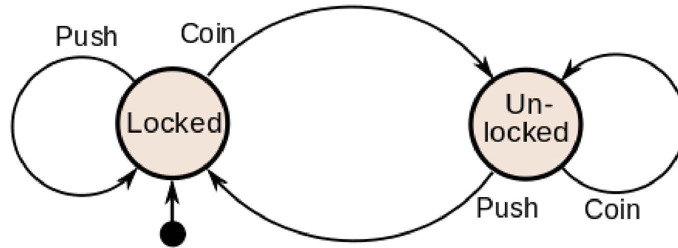


Figure 2.5: finite-state machine example. Turnstile. *The starting state represents the* lock *condition. Inserting a coin generates a* coin *event that triggers the transition from* Lock *to* Unlock. *Pushing generates the* Push *event that triggers the inverse transition and lead back to the start. Image taken from* `http://en.wikipedia.org/wiki/Finite-state_machine`.

It is possible to associate actions to a single state. These action are defined as *entry action* (perfomed when entering the state) and *exit action* (performed when leaving the state, before the beginning of the triggered transition).

The most simple example of finite-state machine is reported in Figure 2.5 and represents a two-states finite-state machine with one transition from the starting state to the ending state (and *vice-versa*).

The formalism used in Figure 2.5 to represent a finite-state machine is called *statecharts* representation and is particularly useful to underline states and transitions.

Finally, one powerful extension to finite-state machine formalism is the possibility to define *hierarchical state machines*. A hierarchical state machine is a UML-based innovation that allows state machines composition with a well-defined hierarchy of states and transitions. Nesting of states is achieved with the following semantics: if a system is in the nested state (*sub-state*) it is also (implicitly) considered to be in the surrounding state (*super-state*). Transition may happen from a super-state to a sub-state one or viceversa, with a variable depth. If an event is not handled by states with the same hierarchy, it is not discarded (like in the traditional finite-state machine formalism) but automatically handled by the state with higher hierarchy (i.e. the super-state) [19]. A simple example is

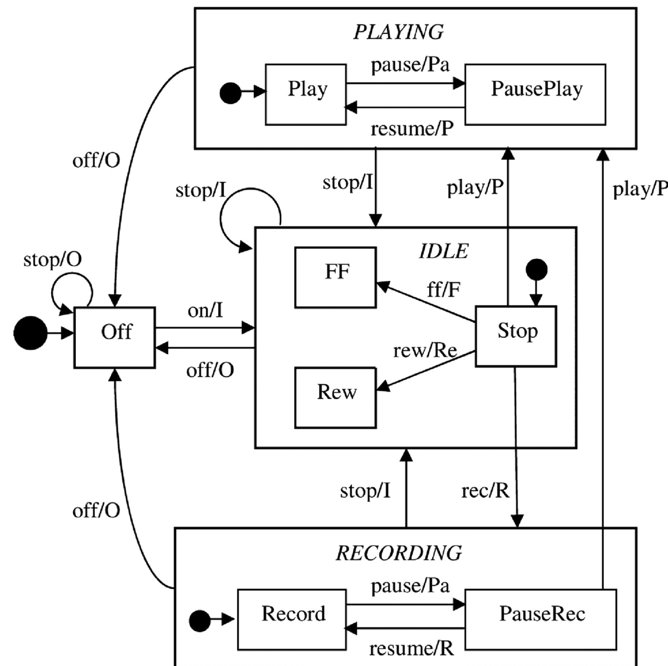


Figure 2.6: Hierarchical Finite-State Machine example. Tape-Recorder. *The higher-level finite-state machine describes transitions between states that are represented by a lower-level finite-state machine. Each lower-level finite-state machine describes its own states and transition. Image taken from `http://comjnl.oxfordjournals.org/content/52/3/334`.*

the finite-state machine representation of a tape-recorder, and it is reported in Figure 2.6. The four main states in which the tape-recorder can be (super-states, i.e. off, idle, playing, recording) are developed in particular sub-states so that each one is actually a finite-state machine itself. The main advantage in using this kind nested of finite-state machines is that if one of the lower-level state machines needs to be modified or substituted, the change does not affect the whole system integrity.

Thus, describing the behaviour of the system through a finite-state machine allows system modularity and extensibility. The definition of hierarchical and innested finite-state machines leads to a powerful flexibility of the controller and guarantees reusability of the single lower-level finite-state machine.

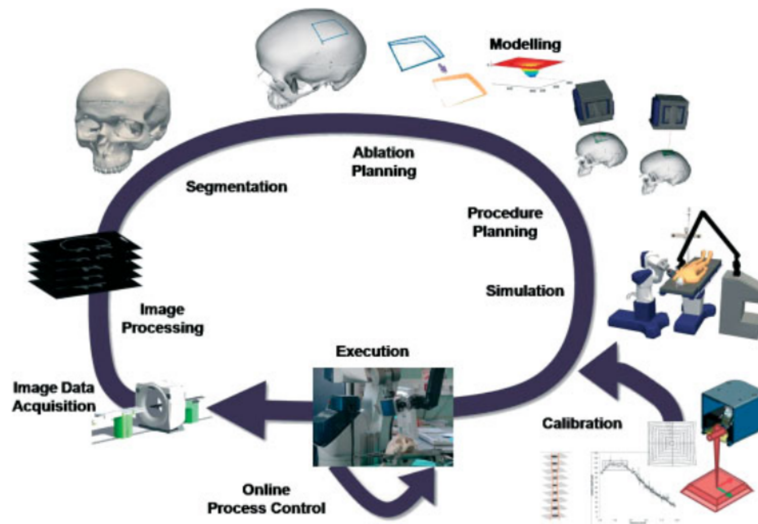


Figure 2.7: Workflow for a laser bone ablation procedure. *Each surgical step is interpretad as a state of a finite-state machine that manages the complete system status. Image taken from [20]*

2.4.2 Examples of finite-state machine based robot behavioural controller

In this paragraph we will provide some examples of robotic system control based of finite-state machine formalism. Because of the aim of the work, we will particularly focus on surgical robotic applications.

Laser Bone Ablation: Surgical Workflow Definition. A particular case of robot behavioural control is the definition of a surgical workflow. In [20] an architecture that orchestrates a robot, a laser and a scanner in order to execute a bone ablation intervention is presented. The complete workflow is represented in Figure 2.7. The surgical steps are interpreted by a finite-state machine that allows the surgeon to enter or exit particular phases of the intervention (i.e. procedure planning, ablation planning, segmentation, image data acquisition, execution). Due to the criticity that can arise during a surgical procedure, a User Interface (thus, the surgeon) is the responsible for triggering transitions from one surgical step to another.

Figure 2.8: Scrub Nurse Workflow. Statecharts representation. *Boxes represents different states and actions performed by the scrub nurse robot. Arrows represents transitions between states and actions. The whole schema represents the complete workflow.*

Scrub Nurse Robot Control. In [21] is presented a robotic system that aims to substitute the role of a scrub nurse during microsurgical interventions. The developed system has to take care of intruments identification, grasping, movement to and from the surgeon's hand, detection of grabbing intentions, obstacole identification and collision avoidance. The behaviour of the robot is modeled using a finite-state machine that acts as a context at the planning level during visual servoing and grasping management. A step-by-step representation of the developed finite-state machine is provided in Figure 2.8. Tests are performed in order to evaluate if the developed scrub nurse is able to change its status (thus, if a finite-state machine transition is triggered) when particular events occurs (e.g. surgeon grabbing the instruments should trigger the scrub nurse to release it).

3 | Materials and Methods

3.1 Hardware description
3.1.1 Robot - KUKA LightWeightRobot 4+
3.1.2 Tracker - NDI Optotrak Certus®
3.1.3 Tracker - NDI Polaris Vicra
3.1.4 Tracker - Microsoft Kinect
3.2 Software tools
3.2.1 ROS and OROCOS
3.2.2 rFSM
3.3 Multi-sensor robot tracking
3.3.1 Tool tracking
3.3.2 Calibration
3.3.3 Sensor Fault Analysis
3.3.4 Multi-sensor control architecture
3.4 Multi-kinect people tracking
3.4.1 People detection
3.4.2 People tracking
3.4.3 Calibration
3.4.4 Multi-kinect control architecture
3.5 Robot Behavioural control
3.5.1 Scenario
3.5.2 Finite-state machine description
3.5.3 Behavioural control architecture
3.6 Application
3.6.1 Complete architecture
3.6.2 Graphical User Interface
3.6.3 Performance tests

3.1 Hardware description

In this section we provide a description of the hardware (Robot and Tracking systems) used in the presented work.

3.1.1 Robot - KUKA LightWeightRobot 4+

The Robot used in the presented scenario is a LightWeightRobot LWR 4+ (KUKA[1], Germany), [22]. Since it aims to to work in proximity (and cooperation) with humans, the robotic arm is designed using a human-like approach in term of weight-to-payload ratio (the goal is 1:1) and performances, with 7 degrees of freedom (thus, a degree of redundancy) that make possible, e.g., an elbow motion while maintaining the pose of the hand.

The main characteristic of the LWR 4+ are listed in Table 3.1, [23]. They include:

- A payload of 7 kg versus a mass of 16 kg. The low mass reduce the power consumption and guarantees the ability of the robot to be manually carried to the place of use.
- 7 degrees of freedom, thus a degree of redundancy.
- Torque and position sensors in each of the seven joints.
- The ability to act like a spring-damper system in which the parameters can be set within wide limits.

[1] http://www.kuka.com/

Type	Lightweight Robot LWR 4+
Number of axes	7
Volume of working envelope	1.84 m^3
Repeatability	$\pm 0.05mm$
Weight	16 kg

Table 3.1: *Lightweight Robot LWR 4+ Basic Specifications*

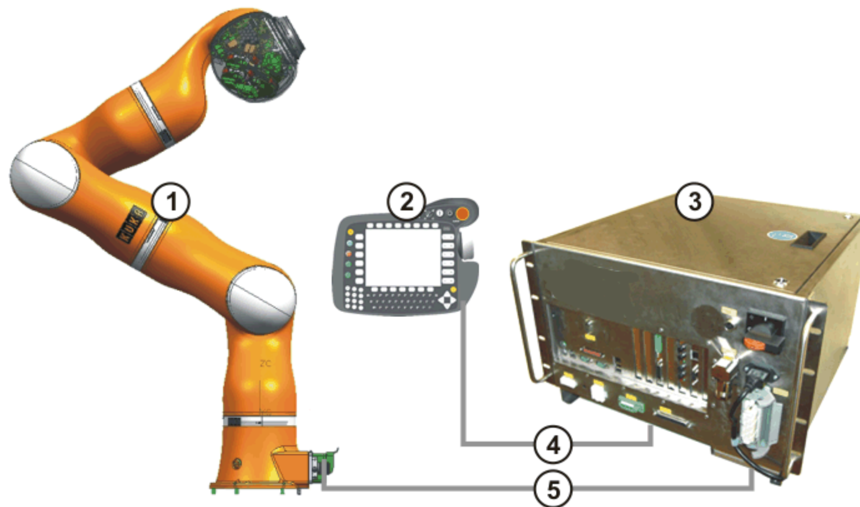


Figure 3.1: KUKA LWR 4+ and Controllers schema. *The LWR 4+ robotic arm (1) is shown with both the TeachPendant (2) and the* KUKA Robot Controller *(KRC, 3). Cables that allows connections between the three elements are shown (4,5). Image taken from [23].*

From the software point of view, the LWR 4+ uses a two-sided controller, a basic controller (developed by German Aerospace Agency[2], Germany) and an estabilished programming and operation environment (the KUKA TeachPendant). The two controllers communicate both asynchronously and synchronously at the interpolation rate of the LWR. The TeachPendant provides user-friendly interface to the robot (running in Windows environment) and a simple programming language to execute scripts in order to obtain simple actions or communicate with more advanced controller running on an external host. The robot, togheter with the controllers, is shown in Figure 3.1.

[2]`http://www.dlr.de/`

3.1.2 Tracker - NDI Optotrak Certus®

The Optotrak Certus® (Northern Digital Inc.[3], Canada) is an optical measurement device able to track the position of infrared Light Emitting Diodes (*markers*) within a specific area. The position sensor consists in three one-dimensional Charge-Coupled Device (CCD) array paired with cilindrical lens and held on a rigid bar at a distance of 1.1 m, pre-calibrated by the manufacturer. The *markers* can be contained in a polymer base with wiring attached or not, with a diameter dimension of 7, 11 or 16 mm. *Markers* are linked to the tracking system through a strober connected to the controller unit. The controller unit takes care of the serial activation of each marker and of the retrieve of sampled data. The complete specifications [24] of the Optotrak Certus® are reported in Table 3.2. The main features are an extremely high sampling frequency (over 900 Hz for 4 markers) and the wide field of view (see Figure 3.2).

General Specifications	
Maximum number of markers	512
Maximum number of rigid bodies	170 (3 markers/rigid body)
Maximum sampling rate (marker frequency)	4600 Hz
Maximum Frame Rate	$\frac{4600}{(N+2)} Hz$
Single Marker Specifications	
Marker voltage (V)	6.0 to 12.0
Duty Cycle (%)	10 to 85
Accuracy Specification	
Volumetric accuracy	0.1 mm
Resolution	0.01 mm

Table 3.2: *Optotrak Certus® specifications.*

[3] `http://www.ndigital.com`

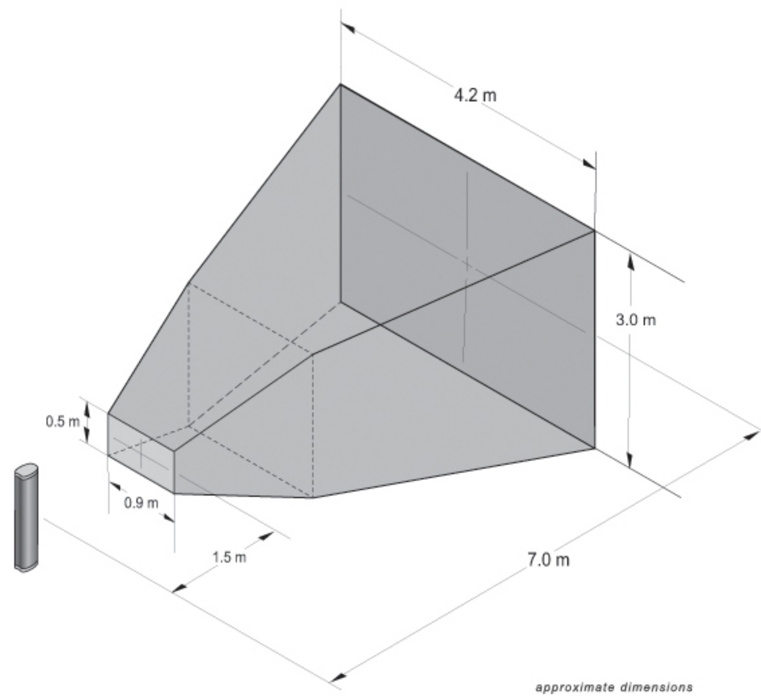


Figure 3.2: Optotrak Certus® Field of View. Image taken from [24].

3.1.3 Tracker - NDI Polaris Vicra

The Polaris Vicra (Northern Digital Inc.[4], Canada) is a tracking system able to provide the 3D position of a set of retroreflective spheres (*passive markers*) when enlighted from an infrared source of light. The Position Sensor emits infrared light from its illuminators (similar to a flash on a conventional camera) that floods the surrounding area and reflects back to the Sensor itself the *passive markers*. The Sensors are two Charge-Coupled Device (CCD) matrices paired with lenses and held on a rigid bar at a distance of 27.3 *cm*. The design of the entire system is aimed to the OR usage and so fulfills requirement of portability and space occupation. This leads to a reduction of the specifications and of the field of view (see Figure 3.4).

The complete set of specifications of the Polaris Vicra are reported in Table 3.3.

[4] `http://www.ndigital.com`

General Specifications	
Maximum number of markers	32
Maximum number of rigid bodies	6
Maximum Frame Rate	20 Hz
Accuracy Specification	
Volumetric accuracy	0.25 mm RMS

Table 3.3: *Polaris® Vicra specifications.*

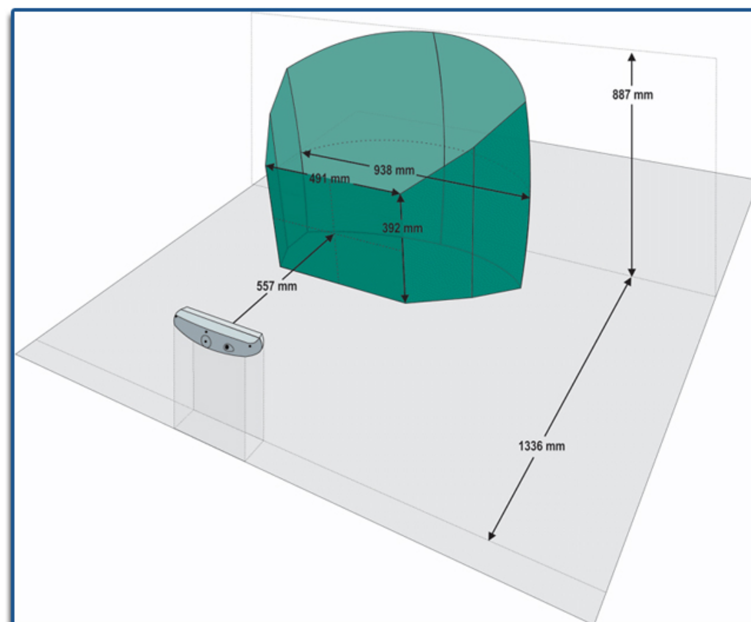


Figure 3.3: *Polaris® Vicra Field of View. Image taken from [25].*

Figure 3.4: Microsoft Kinect® Sensor. *The infrared (IR) projector, the RGB camera and the IR Sensor are showed in figure. Image taken from* `http://www.codeproject.com/Articles/317974/KinectDepthSmoothing`

3.1.4 Tracker - Microsoft Kinect

The Microsoft Kinect® is a sensor able to capture depth and color image simultaneously, at a frame rate of up to 30 Hz. Primarily designed for human interaction in a computer game environment, the sensor showed characteristics of captured data able to find application in many different fields.

The Kinect sensor consists of an infrared laser emitter, and infrared camera and an RGB camera, mounted on a rigid bar and calibrated after factory production. The measurement process is described as triangulation-based [26]. The laser source emits a beam that splits by diffraction into multiple beams, creating a constant well-defined pattern over the scene. The pattern is captured by the infrared camera and correlated against a reference pattern. The reference pattern is constructed with a calibration process and stored into the memory of the sensor. Difference in shape of the scene leads to difference in the captured pattern and thus creates disparity in the correlation results. The sensor is able to process the correlation disparity to create an up to 300,000 points-cloud image containing depth information.

An analysis of accuracy and resolution of the sensor is reported in [27]. Each depth image contains a constant 640×480 pixels, and thus the points density will decrease with increasing distance of the object surface from the sensor. Results show both accuracy and resolution dependant from distance with a quadratic function. At the maximum range of 5 m, the random error of depth measurement reaches 4 cm.

3.2 Software tools

In this section we describe the software tools used to develope the high-level control architecture used in this work. We explain the object-oriented approach used to write components and the tools choosen to manage a stand-alone *task descriptor* (i.e. a finite-state machine) and an automatic deployment of the entire system.

3.2.1 ROS and OROCOS

The software control architecture is developed into the Robotic Operative System (ROS[5]). ROS is a distrubuited framework for the design and development of robot software. It emphasizes the use of modularity, allowing single modules to be individually written and losslessy connected runtime, with a powerful communication management between modules.

Fundamentally, ROS implementation is based on few concepts [28]. **Nodes** represent the basic computational unit (actually they are *software modules* executing an user-defined program). Runtime, *nodes* communicate with each other by passing **messages**, which are strictly typed data structures. A *Message* is defined through very simple Interface Definition Language (IDL) files in order to achieve multi-language support. *Messages* are sent from one *node* to another by publishing them to a given **topic**. *Nodes* can provide data to the architecture publishing them to a *topic* and can retrieve data from a topic simply subscribing to it. Thus, ROS implements a publisher-subscriber protocol. The communication between two (or more) *nodes* potentially can run over different hosts connected in a network and it is managed by a **ROS Master** (i.e. a server) that provides naming and registration services to the ROS system. Data exchange through a *topic* is first initialized by the *Master* and then it works like a strict peer-to-peer communication. A representation of these ROS basics is provided in Figure 3.5.

The main limit of the usage of ROS is in real-time operations. ROS is not

[5] `http://www.ros.org/`

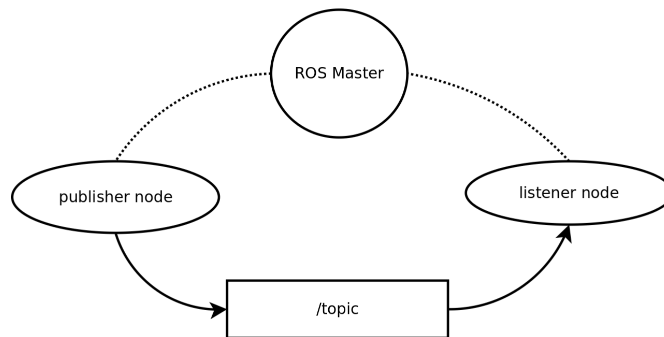


Figure 3.5: ROS basic concepts. *The ovals represent the* nodes *while the box represents a* topic. *The circle represents the server (* ROS Master*) that initialize the communication between the two nodes providing naming and registration services. Data are exchanged between* nodes *through* topics. *In this simple example is reported a single* publisher node *and a single* listener node. *However, there is no limit to the number of* listener *for each topic, even if only one* publisher *is allowed for each topic.*

a real-time framework. However, it exists a particular ROS stack, called Open Robot Control Software (OROCOS[6]) that, if executed as a *ROS Node*, it is able to perform hard real-time operations, even though limited to its environment (i.e. the *ROS Node* in which is executed). Communications with processes extern to that specific *node* looses the real-time guarantee.

Apart from the ability of being executed into a *ROS Node*, OROCOS provides a completely stand-alone development environment, with an object-oriented, components-based policy and an internal managing of connections and data exchange. In fact, it is possible to completely build a control architecture using only the OROCOS tools, without the need of the ROS support. The advantage of using OROCOS as a *ROS Node* is the ability to spread the whole architecture into more than one host, while providing local (i.e. one for each host) real-time operations.

Next, we provide a simple description of how OROCOS works. A complete description of the strategies used to provide real-time communication is far from the aim of this work, but a complete documentation can be found in [29].

OROCOS basic unit is called **component** (or **TaskContext**). It executes one or more (real-time) programs into a single thread. The focus of the whole imple-

[6] `http://orocos.org/`

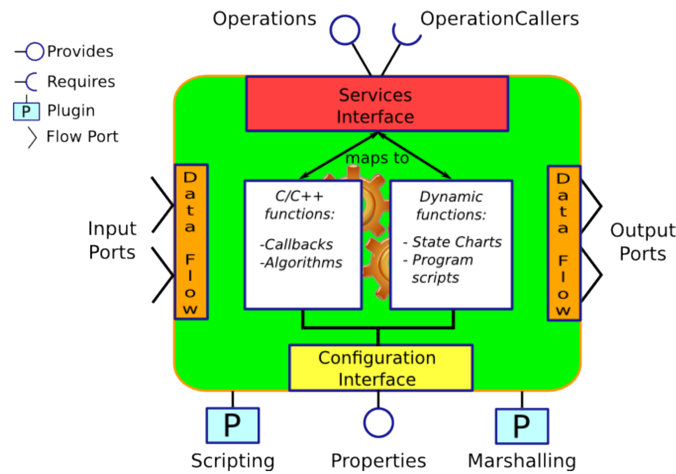


Figure 3.6: OROCOS. Schematic view of a component. *Each component provides a set of user-defined methods (simple algorithm or callbacks) and interfaces to other components and to the environment through data ports (input and output) and* Services *(i.e. methods that can be called from other components). Furthermore, each component can be completely parametric through the usage of properties (actually internal variables that can be modified from other components). Image taken from [29].*

mentation is real-time and thread-safe time determinism. Thus, the system is free of priority-inversion and every operation is lock-free. Each component interfaces the others through **ports** and **services** (i.e. methods that can be managed, thus called, from other components). A complete representation of this basic unit is reported in Figure 3.6. A component is loaded and executed runtime into a deployment manager (called **deployer**, [30]) which takes care of whole architecture setup (e.g. component's configuration, ports connections, etc.).

From the implementative point of view, an OROCOS component is defined as a class that extends the `TaskContext` class. OROCOS is C++ native but, through a particular *deployer* (**rttlua**[7]) and a set of API that takes care of the C++ wrapping, it supports also Lua[8].

Each component is controlled by its *ExecutionEngine*, a finite-state machine that executes pieces of user-defined code in a pre-defined succession (transitions are triggered by the *deployer*). Following a typical object-oriented approach, each

[7] `http://www.orocos.org/wiki/orocos/toolchain/luacookbook`
[8] `http://www.lua.org/`

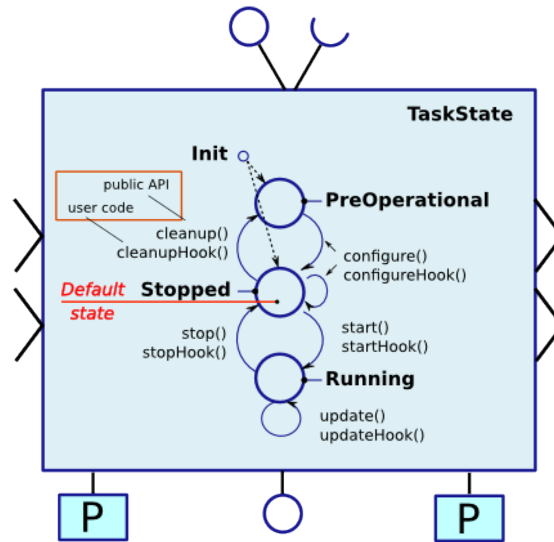


Figure 3.7: OROCOS. ExecutionEngine finite-state machine. *A single component life is identified from its status. First, the component is created in* PreOperational *state by executing its class constructor. Then, the component can be* configured *and* started *to* Running *state, in which it execute a strict-loop program at a certain frequency (i.e. the* update *frequency). This execution can be* stopped*. Error and exception handling are provided by specific states. Each transition executes a user-defined piece of code (called* hook() *program) that performs the wanted computations. Image taken from [29]*

component is first *constructed* (i.e. creation of ports, operations, properties), then *configured* (e.g. setup of user-defined value for properties, creation of connections between ports of different components, etc.). At the end, the component can be *started* (thus, triggered to execute a program in a frequency-defined loop) or *stopped*. This finite-state machine is represented in Figure 3.7. The three main states in which a component may be are called: *PreOperational*, meaning there is a need of further configurations; *Stopped*, meaning that the component is configured and ready to compute; *Running*, meaning that a component has been started.

In this work we design a fully modular control architecture for a surgical robotic system using both ROS and OROCOS in order to mix real-time performance of critical computation togheter with spread of the system over multiple hosts. In particular, the *reasoner* of the entire architecture (i.e. the *high-level control*) is developed completely using OROCOS to achieve the ability or real-time reaction to different events in terms of command sent to the robot. The interfaces to the

rest of the hardware (e.g. the tracking system) is developed using ROS, so that can be executed using a dedicated host for each hardware. Real-time guarantee is then lost regarding the data incoming from sensors.

3.2.2 rFSM

The beahvioural control for a surgical robot developed in this work is based on hieararchical finite-state machines. rFSM, presented by M. Klotzbücher *et al.* in [31], is a general purpose tool for Statecharts (and thus, FSM) implementations written in pure Lua. It is basically a subset of UML representation with simplified execution semantics derived from the STATEMATE statecharts (presented by D. Harel *et al.* in [32]). The basic elements of rFSM are three: **states**, **transitions** and **connectors**.

- A *state* can be a leaf state or a composite state (i.e., a finite-state machine itself) depending on wheter it contains child nodes or not. At the top-level, a finite-state machine is always contained in a state, so that it is easily composable with other state machines. An *entry* and *exit* action can be defined for each state, as well as a *do* function executed (once or in a loop) when the state is active.
- A *transition* is a connection between two states that can define an effect-funcion executed when transitioning.
- A *connector* is used to construct composite transitions by interconnections of two or more elementary ones.

One of the characteristics of rFSM is a strong hierarcy in transitions. In case of multiple triggers (for example when a lower-level state machine's transition is triggered togheter with a top-level state machine one) the higher-level state machine's transition have the priority and thus is executed. In general, the higher the source state of transition is located, the higher the priority of the transition.

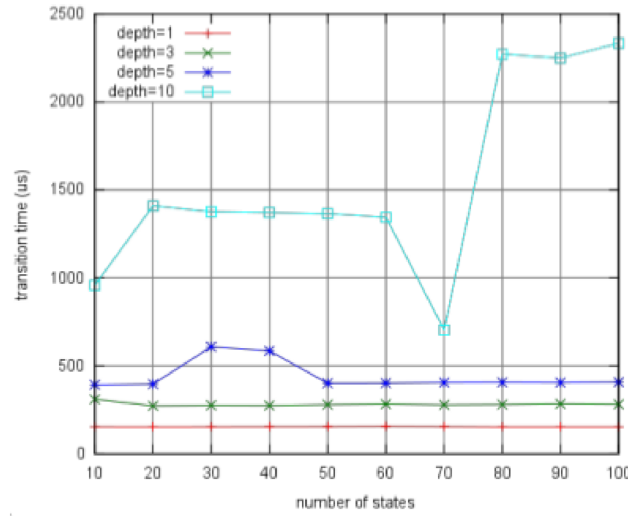


Figure 3.8: rFSM transitions performances. *The figure shows the latencies from a transition trigger to the entrance in the target state. Latencies are strongly dependant from the depth of the requested transition. Transition with a depth inferior to 5 are executed with less than* 500 μs *of latency.*

The main limit of the rFSM tool is the unavailability of a parallel state element. In fact, only one state can be active at a time in a single rFSM instance. Anyway, the usage of a multiple instance of different rFSM can allow a parallel handling of different state machines.

In terms of performances, rFSM time of transition's execution strongly depends on the depth of the required transition, i.e. transition from higher states to lower states requires more time than a transition that lies on the same level. A complete analysis of the performance is reported in [33] and the results are shown in Figure 3.8.

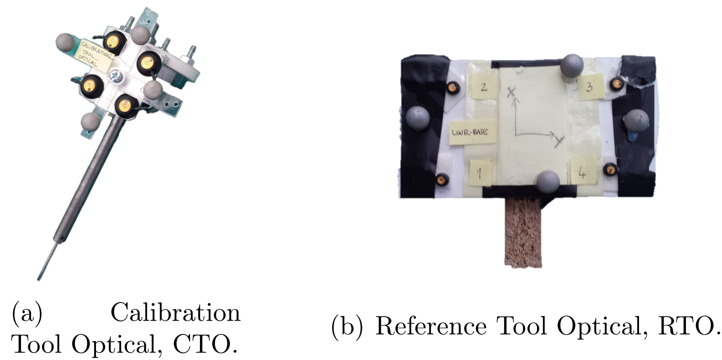
In this work we developed a behavioural control of a surgical robot using a finite-state machine written completely in rFSM and acting like a stand-alone task descriptor for the considered scenario.

3.3 Multi-sensor robot tracking

Intra-operative accurate knowledge of a surgical robot pose is a primary necessity in order to perform accurate end-effector positioning, target following or motion compensation (e.g. movement of the head of the patient during a neurosurgical procedure) and to avoid collisions with the surgical equipe and other instruments. A multi-sensor approach to this task can provide robustness to a single sensor fault, caused for example by a member of the surgical equipe involuntarily covering the sensor field of view. In this section we describe hardware and software strategies used to perform multi-sensor robot tracking during the intra-operative phase of a surgical procedure. A Fault-Tree Analysis for the single sensor is provided in order to study all possible faults that can affect a sensor. Finally, a Software Architecture is presented, with a dedicated component that analizes at each moment all the sensor's status and provides information about the best one able to provide consistent data.

3.3.1 Tool tracking

Standard robot tracking is usually performed by endowing the robot end-effector with a set of well-known recognizable points (i.e. *markers*). Using at least 3 *markers* rigidly constrained it is possible to build a reference frame (*Dynamical Reference Frame*, DRF) that characterizes a rigid-body (*Tool*) with a defined pose in the 3D space. The main problem that arises when switching to a multi-sensor approach of robot tracking is the necessity to provide a *tool* that is capable to be seen from every used sensor. Because of the variability of markers (i.e. active or passive marker with different shape, dimension, reflective properties or control strategies) used by available commercial trackers this appears to be a hard task to accomplish. Moreover, once a multi-sensor visible tool has been provided, the different placement of each sensor (ideally performed in order to cover all the robot workspace) leads to different pose in space of the same *tool* and thus there is the



(a) Calibration Tool Optical, CTO.

(b) Reference Tool Optical, RTO.

Figure 3.9: Developed Optical Tools.

need of a calibration with a common reference.

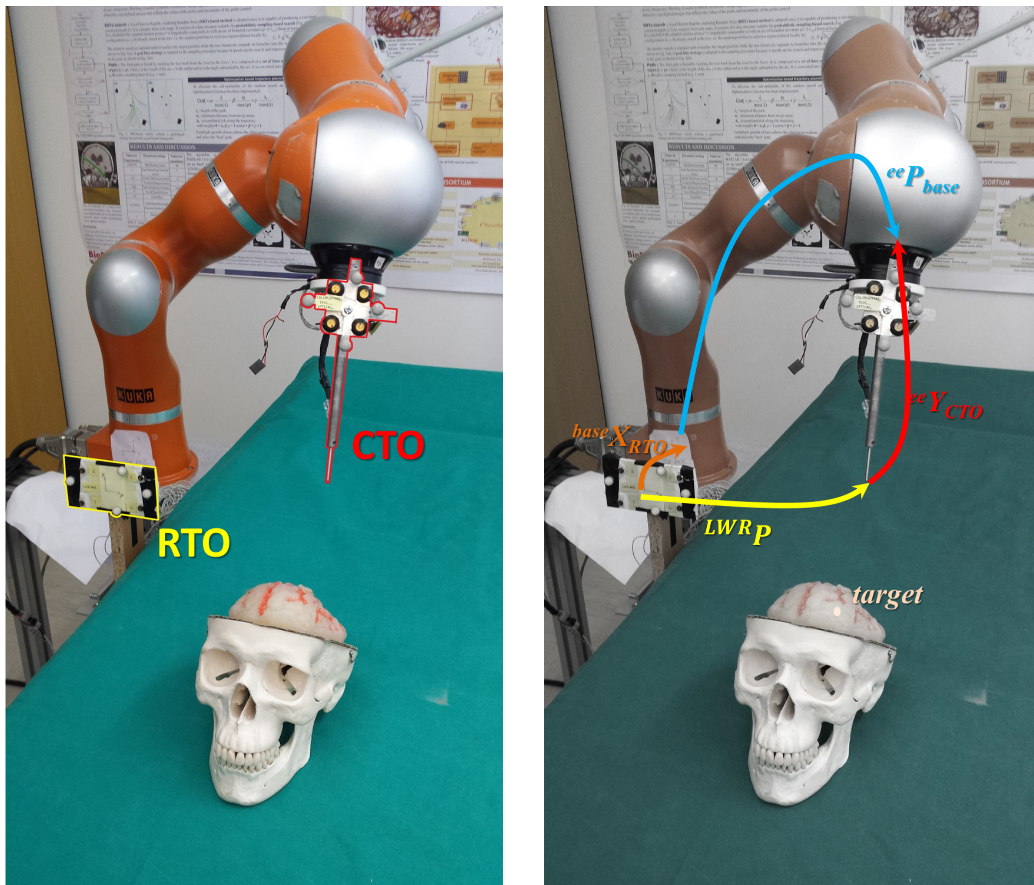
In this work we design an *ad-hoc* multi-marker tool that is able to be seen from both the used tracker and thus perform multi-sensor navigation. The tool is composed by both passive and active markers, arranged in a way that respects the trackers constraint and maintains small dimension and space encumbrance. The markers used are, respectevely, 4 for the NDI Polaris Vicra System (passive) and 4 for the NDI Optotrack Certus System (active). The tool is screwed to the robot end-effector and the *Tool Center Point*, TCP, is localized through pivoting algorithm. This tool (*calibration tool optical*) is represented in Figure 3.9(a).

Following the same approach, we design a reference tool that is capable to be tracked from both the used sensor. This tool (*reference tool optical*, Figure 3.9(b)) is fixed to the operating table and thus represents a common reference for all the trackers.

3.3.2 Calibration

In order to perform intraoperative navigation, it is necessary to calibrate both the single trackers with the robot and the trackers between each other with respect to the *reference tool optical*. All the involved transformations are shown in Figure 3.10(b).

Robot - Tracker Calibration. The calibration between the robot and the single



(a) Calibration Tools. (b) Calibration Transformations.

Figure 3.10: Calibration Tools and involved transformations. *In Figure (a) are represented the calibration tool optical (CTO) and the reference tool optical (RTO) with their position inside the surgical scenario. In Figure (b) the transformations involved during the calibration of the above described surgical tools are highlighted. CTO tool locates the Tool Center Point (TCP) with pivoting algorithms and thus represents the pose of the robot-screwed surgical tool inside the 3D space.* $^{RTO}\mathbf{X}_{base}$ *and* $^{ee}\mathbf{Y}_{CTO}$ *are the transformations that need to be computed for each sensor in order to calibrate the system with an hand-eye approach.*

Calibration Specifications	
Calibration type	*hand-eye*
Number of robot poses	8
Calibration Results - Optotrack	
Translation Residual [m]	[0.0012 0.0025 0.0032]
Rotation Residual [°]	[0.6056 0.5055 0.5412]
Calibration Results - Polaris	
Translation Residual [m]	[0.0018 0.00091 0.0026]
Rotation Residual [°]	[0.7862 1.0609 2.5524]

Table 3.4: *Robot-Tracker calibration specifications and results*

tracker is performed using a classic *hand-eye* approach in which we measure the pose of the *calibration tool optical* (CTO) placed at the end-effector of a LWR4+ robot, ${}^{CTO}\mathbf{T}_{sensor}$, the pose of the *reference tool optical* (RTO) screwed to the operating table, ${}^{RTO}\mathbf{T}_{sensor}$, and the joint configuration of the LWR4+ (from which we can obtain the transformation between the LWR base and the end-effector, ${}^{ee}\mathbf{T}_{base}$) for a number of well-defined robot poses. The calibration transformations able to close the transformation loop, thus

$$ {}^{RTO}\mathbf{X}_{base}, \quad {}^{ee}\mathbf{Y}_{CTO} \tag{3.1} $$

are computed with a least-square minimization algorithm. The chosen number of robot poses is 8 for each sensor and the results of the performed calibration are reported in Table 3.4.

Tracker - Tracker Calibration. The calibration between the two used trackers is performed computing the transformation between $RTO^{Polaris}$ and $RTO^{Optotrack}$, which completely characterizes the two trackers relative poses. Starting from the above described Robot - Tracker calibration, we obtained, respectevely

$$ {}^{RTO,Optotrack}\mathbf{X}_{base}, \quad {}^{ee}\mathbf{Y}_{CTO,Optotrack}, \quad {}^{RTO,Polaris}\mathbf{X}_{base}, \quad {}^{ee}\mathbf{Y}_{CTO,Polaris} \tag{3.2} $$

and thus, we can compute the transformation between the two trackers (${}^{Optotrack}\mathbf{C}_{Polaris}$) as following:

$$\boxed{{}^{Optotrack}\mathbf{C}_{Polaris} = {}^{RTO,Optotrack}\mathbf{X}_{base}({}^{RTO,Polaris}\mathbf{X}_{base})^{-1}.} \tag{3.3}$$

In the above calibration, we chose the NDI Optotrack Certus as the Master Sensor (i.e. the sensor to which the rest is calibrated) because of its higher specifications with respect to all the other used sensors. A generalization of the calibration transformation reported in Equation 3.3 for a non-defined number of sensor can be written as ${}^{Optotrack}\mathbf{C}_i$, where i represents the current sensor returning the pose of the robot. It is of course valid that

$${}^{Optotrack}\mathbf{C}_{Optotrack} = \mathbf{I}, \tag{3.4}$$

where $\mathbf{I}$ represents the identity matrix. Thus, the pose of the LWR4+ in the 3D space and referred to the master *reference tool optical* can be easily obtained at every moment as

$$\boxed{{}^{LWR}\mathbf{P} = ({}^{Optotrack}\mathbf{C}_i)^{-1} \cdot ({}^{RTO}\mathbf{T}_i)^{-1} \cdot {}^{CTO}\mathbf{T}_i, \qquad i = \text{current sensor.}} \tag{3.5}$$

3.3.3 Sensor Fault Analysis

In order to take the maximum advantage from a multi-sensor architecture, it is necessary to exactly know, in every moment, if a specific sensor is able to provide the correct data or not.

Fault-Tree Analysis (FTA) is the most common state of the art in fault analysis [34]. It is used to check the fault propagation on the overall system and to point out the primary reason of a single fault. Each leaf on the tree represents a component failure, and they are combined with a AND/OR logic to build up the upper (i.e. mainly undesired) failure.

During intra-operative robot tracking, the most undesired failure is the incapability of a sensor to provide the correct pose of the tracked robot. This failure can be due to:

- *external causes*, i.e. someone or something occluding the sensor Field Of View (FOV);
- *overpass of techincal specifications*, which usually means that the robot (i.e. the tool) exited the sensor FOV;
- *software failures*, i.e. exceptions raised in the software interface that handles the communication with the sensor's hardware and provides the Control Architecture with the tracking data;
- *hardware failures*, e.g. unexpected power-supply cut, low-level hardware errors, etc.
- *calibration failures*, i.e. the calibration tools were moved from the position they had when the calibration was performed.

External causes, as well as the tool exit from the sensor FOV, leads the architecture to be provided with the information that the tracked tool is MISSING. A particular case must be considered: the positioning of the fixed *reference tool optical* is done *a priori* and taking into account the sensors FOV. Thus, the RTO exit from the sensor field of view is a possibility that can not happen: RTO-related failures are linked only to external causes.

Software exceptions in the interface application lead the component to stop publishing new tracking data. Then, the system will continue to read old tracking data. Hardware failures (e.g. unexpected power-supply cut) lead to the same consequencies.

Calibration-related failures means that the calibration tools were moved from their original position, leading the trackers to provide a wrong robot pose in the intra-operative space.

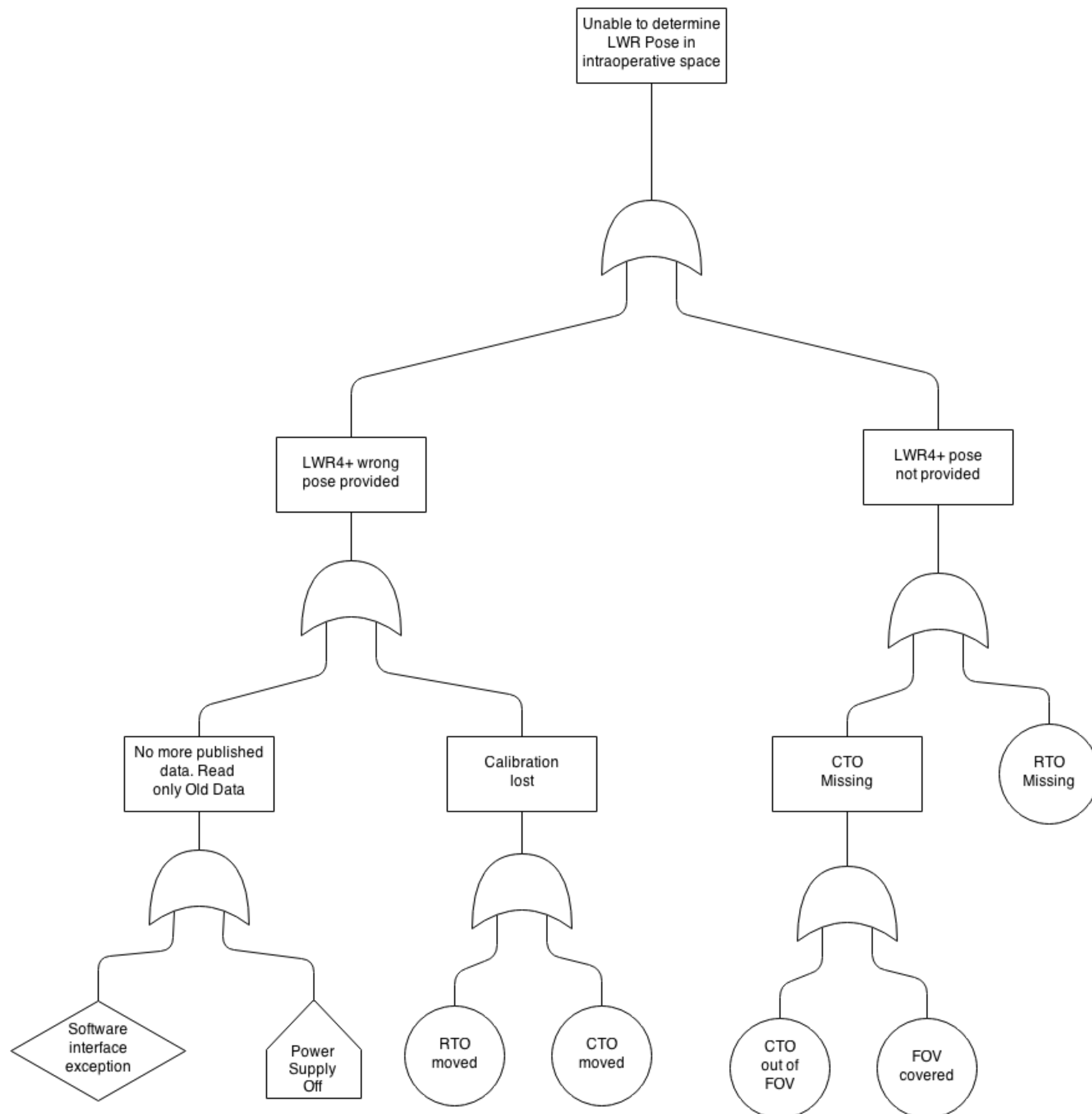


Figure 3.11: Sensor Fault-Tree Analysis. *Causes due to external factors and to the overcome of system specifications (reported to the right side of the tree) lead the architecture to be provided with the information that the tracked tool is MISSING. RTO-related failures are only to be linked to external causes (since RTO is positioned a priori). Software exceptions (reported to the left of the tree and not completely exploded into their causes) lead the component to stop publishing new tracking data. The system then will continue to read old tracking data. Hardware failures lead to the same consequences. Calibration-related failures (at the left of the tree) means that the calibration tools were moved from their original position and so the trackers provide a wrong robot pose in the intra-operative space.*

Starting from these considerations, we designed a Fault-Tree that is reported in Figure 3.11 and represents the link between all the identified causes and the final main failure. The FTA is the starting point to develop a software achitecture able to analize tracking data run-time, providing information about the trackers able to correctly follow the robot movement into the intra-operative space and about which one of these trackers is the most performant one.

3.3.4 Multi-sensor control architecture

Next, we provide a description of the software architecture developed in order to manage a multi-sensor intra-operative surgical robot navigation. Starting from the FTA presented above, the information that a similar architecture has to provide are reported below, togheter with other information that will enhance the security-oriented approach we follow:

- The ability of each sensor to correctly track the robot at a specific moment;
- Which of the available tracking sensors is the most performant one;
- The calibration information about the best available sensor.

Moreover, since every sensor has its own specifications (i.e. accuracy, sampling frequency, field of view) the definition of the robot velocity of movement should be done according to the current sensor used for tracking. Thus, the architecture also should provide

- The maximum velocity and acceleration that the robot can use when moving tracked by the current best available sensor

to manage the different characteristics of the used sensors. A UML diagram of the developed architecture is represented in Figure 3.12. It is composed by a set of components that interfaces the hardware sensor (two in our case scenario, one for

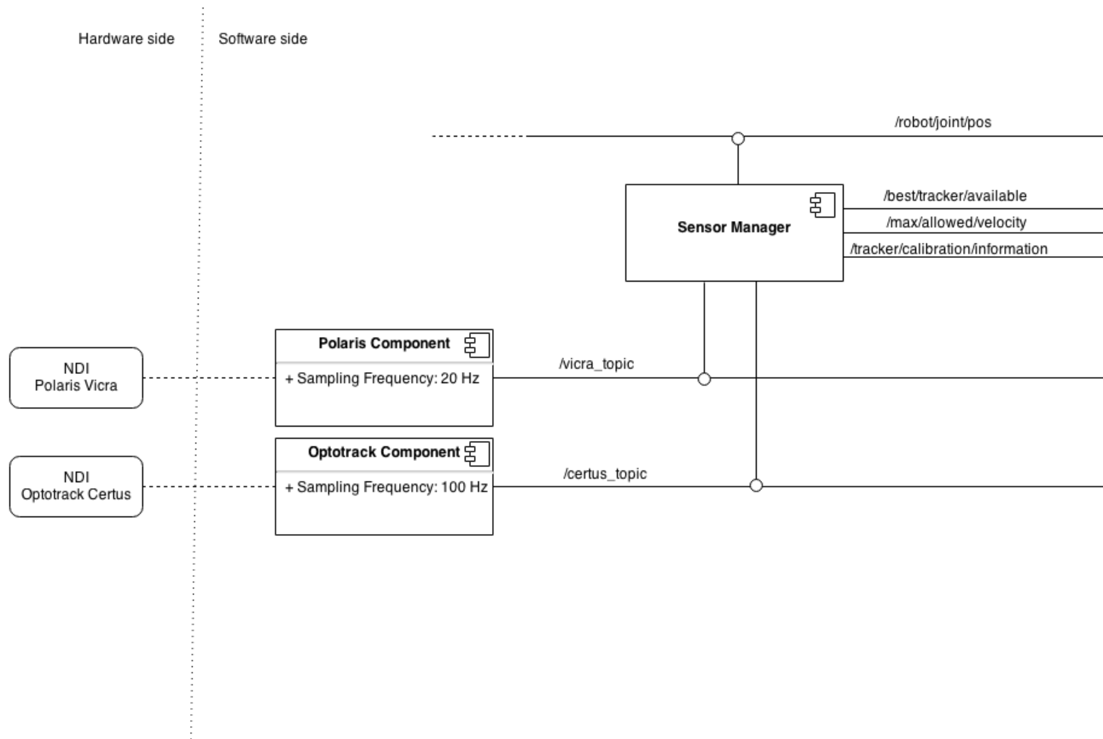


Figure 3.12: Multi-sensor tracking architecture. *The rounded boxes (i.e. NDI Polaris Vicra and NDI Optotrack Certus objects) represent hardware components. Dashed lines identifies communication protocols different from the ROS or* OROCOS *communication protocol. The* Polaris Component *and* Optotrack Component *are the developed software interfaces that handle communication with the sensor hardware and publish the tracking data over a set of dedicated ROS topic. The* Sensor Manager *component is the reasoner of the architecture and analizes the incoming data, providing information about the status of the sensor and their ability in tracking the robot.*

the NDI Polaris Vicra, called **Polaris Component**, and one for the NDI Optotrack Certus, called **Optotrack Component**) and a *reasoner* component called **Sensor Manager** that analizes the incoming data and provides the information stated above.

Polaris Component. The *Polaris Component* is a ROS node that handles the communication with the NDI Polaris Vicra System and streams the tracking data on a set of dedicated *topic*s.

The communication with the Polaris System takes place using a USB cable interfaced though the Future Technology Devices International (FTDI) drivers[9], thus working the same way of a serial communication. A complete set of C++ Application Programming Interfaces (APIs) was developed in this work in order to allow communication with the tracking system.

The geometry information about the different Tools to be tracked ($RTO^{Polaris}$ and $CTO^{Polaris}$) are represented as binary file (`.rom`) generated using the Architect 6D software (NDI, version 3, 2005) and loaded at startup into the system. No constraint to the number of Rigid Bodies that can be tracked is imposed from the software side (the limit is the Polaris System own limit about the maximum number of tools that can contemporary be tracked) and the addition of one more tool only requires the modification of a `.xml` file that contains the path to the new `.rom` file, without the need to re-compile the code.

The update rate for every tool is 20 Hz. Each tool's tracking information are streamed in the form of position vector and quaternion of orientation over a topic labeled `/vicra/name_of_the_tool` and thus available to all the different hosts connected to the same ROS network.

Optotrack Component. The *Optotrack Component* is a ROS node that manages the communication with the NDI Optotrack System and streams the traking information over a set of dedicated ROS *topic*. The communication with the Optotrack Component takes place over an ethernet cable and is managed using the Northern Digital Inc. provided APIs.

The geometry information of the tools to be tracked ($RTO^{Optotrack}$ and $CTO^{Optotrack}$) are contained in binary files (`.rig`) generated using the Architect 6D software

[9] `http://www.ftdichip.com/FTDrivers.htm`

(NDI, 2006). These files are loaded at startup of the application into the Optotrack controller and the addition of more `.rig` files requires only the modification of a `.xml` file containing the path to the binary file.

The update rate for every tool is 100 Hz and each tool's tracking information are streamed over a dedicated ROS *topic* (called `/certus/name_of_the_tool`) in the form of position vector and orientation quaternion.

Sensor Manager Component. The real core of the multi-sensor architecture is represented by the *Sensor Manager* component. This is an OROCOS TaskContext that reads the data coming from trackers and chooses at each moment the best available tracker with respect to the capability of the single sensor to track both the robot (i.e. the CTO) and the intra-operative reference frame (i.e. the RTO) and the sensor accuracy in providing tracking data.

When instatiated, the component creates a set of `InputPort` that can read the data incoming from the different sensors. An additional (and optional) `InputPort` is created and, if connected, provides the component with the ground truth about the robot position (i.e. the robot joint configuration). Both the data regarding the RTO and the data regarding the CTO are analized run-time with an update rate of 1 kHz (chosen in order to be faster than the most perfomant sensor connected).

The number of the connected sensors and the knowledge needed to reason about data consistency are stored in a `RTT::PropertyBag` and loaded at startup from a `.cpf` file[10] using OROCOS marshalling service. `InputPort` are created following the information reported in this file thus, there is no limit to the number of sensors that can be connected to the architecture without re-compiling the code. The complete `RTT::PropertyBag` contains the following fields for each sensor:

- `sensor priority`. We defined a strong sensor hierarchy based on the tracker's specifications (i.e. sampling frequency, field of view, accuracy).

[10] A `.cpf` file has the same sintax of an `.xml` file.

- `calibration data`. The ${}^{RTO,sensor}\mathbf{X}_{base}$ and ${}^{ee}\mathbf{Y}_{CTO,sensor}$ matrix.
- `allowed velocity`. The velocity that the robot can use when moving tracked by this sensor.
- `allowed acceleration`. The acceleration that the robot can generate when moving tracked by the current sensor.

Once loaded, the data are stored in a struct and sorted by the sensor priority. Each analysis is thus performed starting from the tracker with the highest priority (i.e. the most desiderable one) to the tracker with the lowest priority (i.e. the less desiderable one).

The performed analysis is based on the Fault-Tree Analysis presented in the above paragraph (see 3.3.3) and is described in a flowchart reported in Figure 3.13.

We defined a strong hierarchy for the fault that can happen to the single sensor in order to optimize the number of performed computations. Referring to Paragraph 3.3.3, *external causes* and *overcome of technical specifications* are the fault with the highest priority because they don't provide a robot pose in the 3D space at all. After those causes, *hardware failures* are considered with higher priority than *calibration failure*. Once a fault with a defined priority is detected, the sensor failure is asserted without the need to check for the other fault causes.

On the other side, when a sensor that has a defined priority with no failure is detected (and thus, a sensor that can correctly provide the robot pose) the analysis is ended without checking the sensors with lower priority.

The results of the analysis are published over 3 `OutputPort` respectevely regarding: (1) the most performant tracker actually correctly tracking the robot, (2) and (3) knowledge about that specific tracker (i.e. robot allowed velocity and acceleration, calibration data).

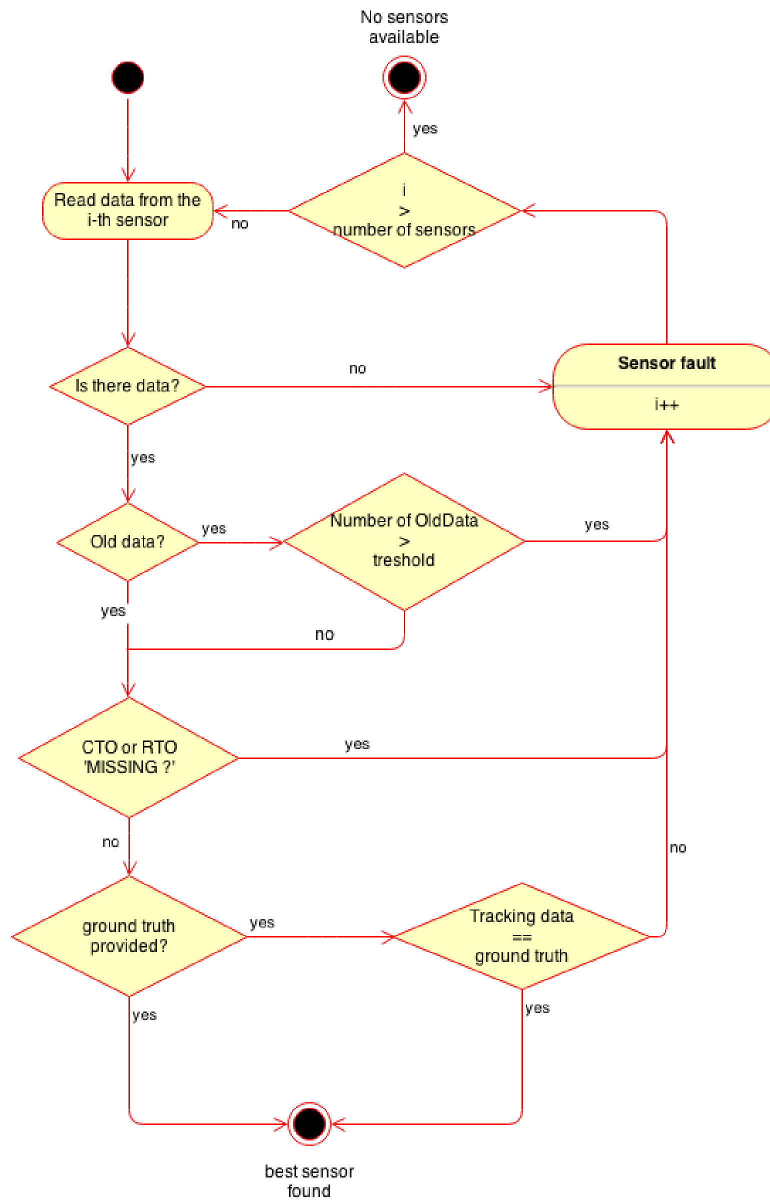


Figure 3.13: Sensor Manager. Data Analysis Flowchart.

3.4 Multi-kinect people tracking

As already stated, ORs are crowded environment in which a surgical robot needs to move without collisions with object and, in particular, people belonging to the surgical equipe. Depth-based imaging (for example, using Microsoft Kinect) can provide a simple way to handle this task. Moreover, a multi-sensor approach is robust to failures due to the impossibility to recognize and track people covered from surgical instrumentation (e.g. the robot itself) or from other members of the equipe. Here we describe the software approach used in this work to perform multi-Kinect people detection and tracking inside the OR. A software architecture that handles computations is presented in order to provide the system with the data regarding the people detected inside the OR.

3.4.1 People detection

Regarding people detection, we refer to the task of automatically recognising a human body inside a depth image. Given a sensor able to return a point-cloud image containing the depth data, there are software libraries able to recognize user's body run-time, starting from the set of provided data. A widely used solution for Microsoft Kinect is the open-source combination of the OpenNI framework[11] and the NITE libraries[12]. OpenNI (PrimeSense, Israel) is the open-source version of the SDK software developed by Microsoft to handle Kinect data. These tools provide a complete set of Application Programming Interfaces (APIs) able to manage the whole set of data coming from the Kinect sensors and also to perform people detection. Up to 16 users can be detected at the same time for a single Kinect sensor. Moreover, they are completely integrated inside ROS environment through the `openni_camera` and the `openni_tracker` stacks[13].

We performed people detection for each sensor using a modified version of the

[11] `https://github.com/OpenNI/OpenNI`
[12] `http://developkinect.com/tags/nite`
[13] `http://wiki.ros.org/openni_camera`

`openni_tracker` stack (`kinect_tracker`) that is able to manage more than one Kinect sensor on the same host and communicate the number of connected sensors to the whole ROS network using a dedicated parameter on the ROS Parameter Server (i.e. `number_of_connected_kinects`).

People detection is achieved without the need of a calibration pose, because of the possibility in OpenNI (v. 1.5.2) to trigger a callback when a point cloud inside the image is recognized as a human skeleton.

3.4.2 People tracking

Using OpenNI, after a person (*user*) is recognized by the measurement system, his position in the 3D space is returned as the pose of a set of frames having their origin in the joints of that specific person. As reported in Paragraph 3.1.4, accuracy of detection is low. Thus, particular security ranges must be considered when checking for collisions. The complete set of frames constructed for each user is reported in Figure 3.14. The main problem when switching to a multi-sensor approach is merging data incoming from the different sensors. In the simple case of only one person inside the tracked environment, the users recognized from each sensors (i.e. up to one for each sensor) are actually the same person and the architecture needs to be able to find this correspondence.

In the case of one user and two sensors, we define the 3D pose in the i-th sensor reference frame of each joint as:

$$^{sensor}\mathbf{P}_{joint}, \tag{3.6}$$

where *sensor* refers to the Microsoft Kinect System that measures the specific user while *joint* refers to the joint of human body, the data that we obtain when the user is tracked by both the sensors are:

$$^{1}\mathbf{P}_{joint}, \qquad ^{2}\mathbf{P}_{joint}, \qquad \text{where} \qquad joint = 1, 2, \ldots, 15. \tag{3.7}$$

head	
neck	
torso	
left shoulder	right shoulder
left elbow	right elbow
left hand	right hand
left hip	right hip
left knee	right knee
left foot	right foot

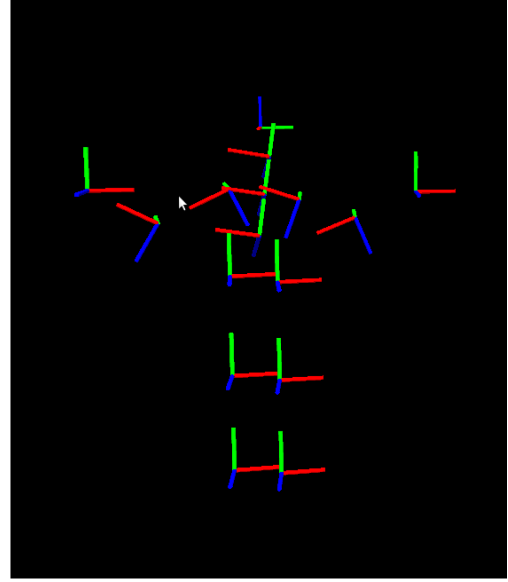


Figure 3.14: User's joint reference frames. *For each detected users, 15 reference frames are constructed for 15 different joints. The joints are reported at the right side of this figure in a Table. Their 3D pose in space is returned in order to perform tracking. A user currently tracked is reported, with its own reference frames, at the left side of this Figure.*

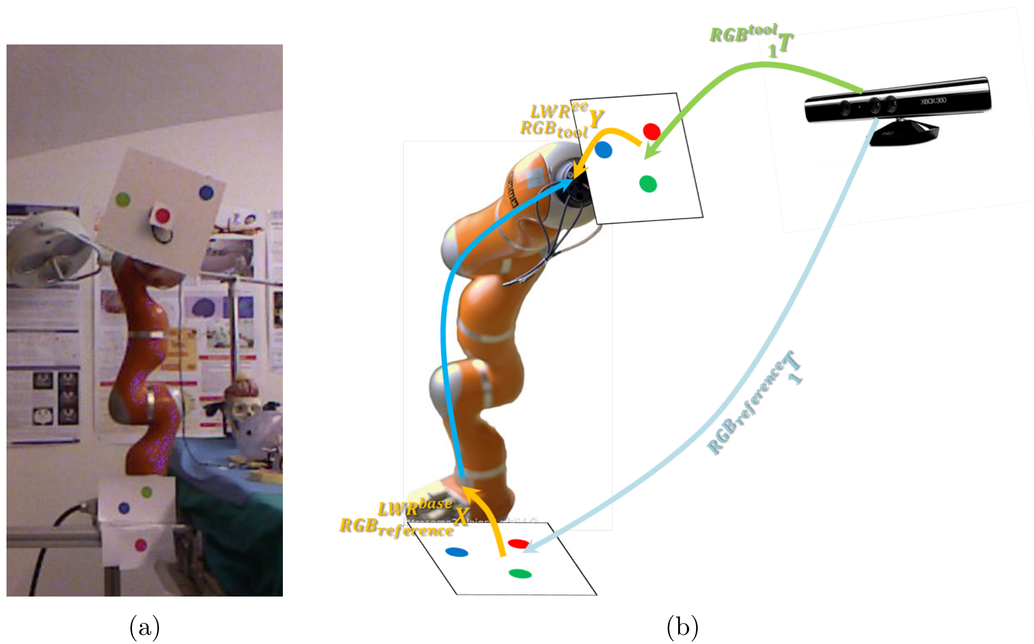
Considering the 1st Kinect as the Master Sensor (i.e. the sensor to which the other sensors are referred) and since ${}^{1}\mathbf{T} = {}^{1}\mathbf{C}_2 \cdot {}^{2}\mathbf{T}$, where ${}^{i}\mathbf{T}$ is the reference frame of the i-th sensor and ${}^{1}_{i}\mathbf{C}$ is the calibration transformation between the i-th sensor and the Master Sensor, then if

$$ {}^{1}\mathbf{P}_{joint} = {}^{1}\mathbf{C}_i \cdot {}^{2}\mathbf{P}_{joint}, \qquad \forall \text{ joint} \tag{3.8} $$

the user tracked by sensor 1 and the user tracked by sensor 2 are actually the same user. A more realistic condition that considers noise, accuracy of the sensors and error of calibration can be written using the euclidean distance between two users, thus

$$ \boxed{d({}^{1}\mathbf{p}_{joint}, {}^{1}\mathbf{C}_i \cdot {}^{i}\mathbf{p}_{joint}) < T, \qquad \text{sensor} = 1, 2, \ldots, n \qquad \forall \text{ joint}} \tag{3.9} $$

where $d(\cdot,\cdot)$ represents the euclidean distance, ${}^{1}\mathbf{p}_{joint}$ represents the position vector of the user in the Master Sensor reference frame, ${}^{1}\mathbf{C}_i \cdot {}^{i}\mathbf{p}_{joint}$ represents the



(a) (b)

Figure 3.15: Calibration Tools and involved transformations. *The two used RGB tools, RGB^{tool} and $RGB^{reference}$, are represented in (a) with their position with respect to the LWR robot during the calibration procedure. The involved transformations are represented in (b). Photo on the left is reported with courtesy from A. Antonietti (M.Sc., Politecnico di Milano) and A. Cingolani (Politecnico di Milano). Image on the right has contributions from M.D. Comparetti (Ph.D.).*

position vector of the user transformed from the i-th sensor reference frame to the Master Sensor and T represent an arbitrary treshold. Considering accuracy of the Kinect sensor (See Paragraph 3.1.4) and the errors of calibration, we chose a threshold of 45 cm. This conditions only refers to position and does not consider the orientation of the user's joint. The Equation 3.9 represents the condition used in this work to discriminate between users detected from different cameras in the same workspace.

Once a user is detected and correspondences have been found, a simple envelope is built, with spheres and cylinders constructed along the detected skeleton.

3.4.3 Calibration

The calibration matrix ${}^{1}\mathbf{C}_{i}$ used in this Thesis is computed using the calibration procedure presented by T. Beyl *et al.* in [35]. Corners on a 640 $\times$ 560 mm

chessboard (5×6 squares) are identified using an OpenCV algorithm and used to perform a correspondence points calibration between two Kinect sensors. The placement of the two sensors is accomplished in order to avoid blind-points in the overall FOV, with the Master Kinect Sensor fixed to the ceiling. 4 different poses of the chessboard in a 2×4 m workspace are used to compute the pairwise calibration matrix between the i-th sensor and the Master Sensor.

Once the used sensors are calibrated between each others, in order to have information of the users pose with respect to the robot, thus perform collision avoidance or raise emergency events in case an user is too close to the robot during movement, there is the need to calibrate the Master Kinect Sensor with the robot. Using RGB markers (RGB^{tool} and $RGB^{reference}$, see Figure 3.15(a)) placed both at the robot base and end-effector, a standard *hand-eye* calibration is performed in which we computed the ${}^{LWR_{base}}\mathbf{X}_{RGB_{reference}}$ and ${}^{RGB_{tool}}\mathbf{Y}_{LWR_{ee}}$ trasformations between the calibration RGB tools and the robot kinematics. All the involved transformations used to calibrate the different sensors are shown in Figure 3.15(b). Given these two transformation and the pose of the $RGB^{reference}$ tool (${}^{RGB^{reference}}\mathbf{T}_1$) due to the fixed placement of the Master Kinect Sensor, the pose ${}^{joint}\mathbf{U}$ of a user in the LWR reference frame can be computed as:

$$\boxed{\mathbf{U}_{joint} = {}^{LWR_{base}}\mathbf{X}_{RGB_{reference}} \cdot {}^{RGB^{reference}}\mathbf{T}_1 \cdot {}^{1}\mathbf{C}_i \cdot {}^{i}\mathbf{P}_{joint}, \qquad \forall \text{ joint}} \tag{3.10}$$

where ${}^{1}\mathbf{C}_i \cdot {}^{i}\mathbf{P}_{joint}$ represents the pose of a user detected by the i-th sensor and registered in the Master Kinect Sensor reference frame. It is of course valid that ${}^{1}\mathbf{C}_i = \mathbf{I}$ if $i = 1$, where $\mathbf{I}$ is the identity matrix.

3.4.4 Multi-kinect control architecture

In this section we provide a description of the software architecture developed in order to manage people detection and tracking with a multi-kinect approach. The aim of this architecture is to provide information about the pose of a certain

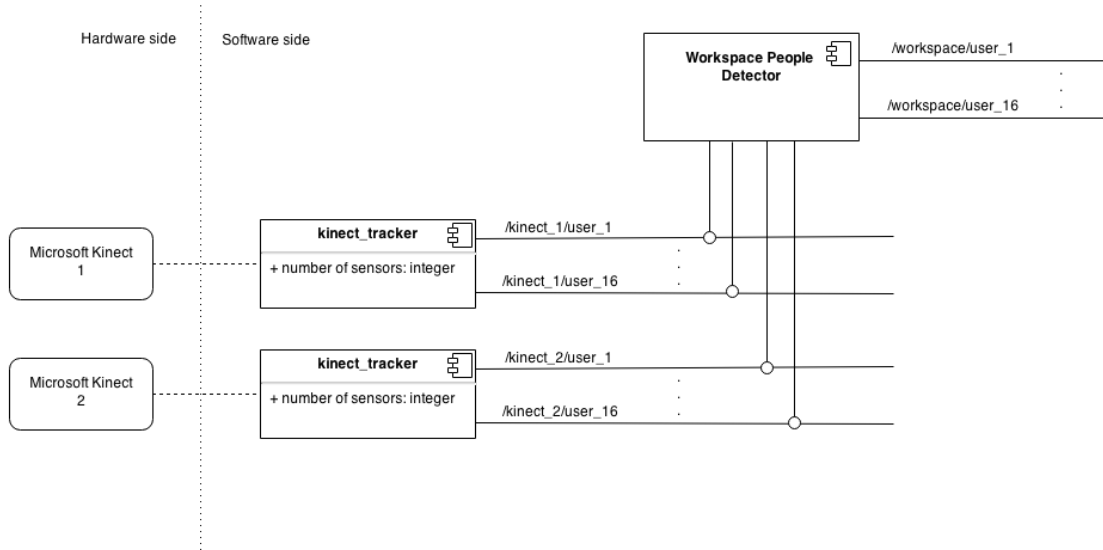


Figure 3.16: Multi-sensor people tracking architecture. *The rounded boxes represent hardware components. Dashed lines identifies communication protocols different from ROS and* OROCOS *one. The* kinect_tracker *component handles the communication with the Microsoft Kinect and performs human detection. The* Workspace People Detector *merges data incoming from the different sensors and returns users data to the system.*

number of users in the considered workspace and referred to the base of the LWR Robot, merging data coming from a variable number of sensor calibrated between each others. The presented architecture is made of a set of different components. Some components (i.e. the **Kinect Tracker** components) take care of the hardware interface and provide single sensor's information to the architecture *reasoner*, the **Workspace People Detector** component, that analizes and merges the incoming data. The UML representation of the developed architecture is reported in Figure 3.16.

Kinect Tracker Component. The *Kinect Tracker* component is a ROS node that handles the communication with one Microsoft Kinect sensor using the OpenNI API. This component is a modified version of the `openni_tracker` package, released as a ROS stack by T. Field. The communication with the Microsoft Kinect takes place using USB protcol.

Every time the component is started, it reads a dedicated parameter inside the

ROS Parameter Server (i.e. `number_of_connected_sensors`) and updates the parameter with the information about the new sensor that is connected to the ROS network. Acting this way, the complete architecture is always able to have knowledge about the number of connected sensors and to retrieve the required data from the Parameter Server. After the advertisement that about a new sensor connection to the whole architecture, the component instatiates a OpenNI callback that is triggered when a human skeleton is recognized inside the depth image. The callback takes care of reading the pose of the reference frames constructed upon the user's joints and to stream the data over a set of dedicated ROS topic. A custom set of ROS messages (i.e. `kinect_msgs/`) is created to manage the communication of the Kinect data inside ROS environment.

Workspace People Detector. The *Workspace People Detector* component is a ROS node that reads the incoming data from each sensor, merging the different users information to provide the pose of a set of users (up to 16) inside the robot base reference frame. Once the node is started, it retrieves information about the number of connected sensors from the ROS Parameter Server and accordingly instatiates a set of subscribers and callbacks in order to read the incoming data for each Kinect sensor. There is no theoretical limit to the number of Kinect sensors that can be connected to the architecture without the need to re-compile the code. Other information needed to discriminate between users detected by different sensors are:

- *sensor-sensor calibration data*, thus the transformation matrix ${}^{1}_{i}\mathbf{C}$ that register the i-th sensor reference frame with respect to the Master Sensor reference frame;
- *robot-sensor calibration data*, thus the tranformation matrix ${}^{LWR_{base}}\mathbf{R}_1 = {}^{LWR_{base}}\mathbf{X}_{RGB_{reference}} \cdot {}^{RGB^{reference}}\mathbf{T}_1$ that registers the Master Sensor reference frame with respect to the robot base.

These data are loaded at startup using a `roslaunch` script that reads parameters from a `.xml` file. The calibration data are assigned to the single sensor instance accordingly with the sensor tag: thus, the sensor with tag '1' is considered as the Master Sensor, and so on.

After the initialization procedure, the component starts the incoming data analysis with a full-parallel algorithm. Triggered callbacks are stored in a queue with First-In-First-Out (FIFO) logic and released (thus, executed) at a specific frequency.

New data from a specific sensor is registered into the Master Kinect Sensor reference frame using the transformation matrix ${}^{1}\mathbf{C}_i$ loaded at startup.

For each new data incoming from a sensor, a comparison with data provided from all the other connected sensor is performed in order to find correspondences between users. If a couple of users, one detected from the i-th sensor and one detected from the j-th sensor, respects the condition stated in Equation 3.9 they are considered as a single user and the data are streamed over the assigned ROS topic. If a sensor does not provide new data at that specific iteration, the comparison is not executed. Finally, if a sensor provides data regarding a user that does not respect the condition in Equation 3.9 when compared with all the other sensors, that specific user is considered as a stand-alone user and its data are streamed over a ROS topic. Multi-threading and parallelism is achieved using OpenMP libraries[14], that split the analysis of data coming from a couple of sensors over different cores of the CPU (i.e. over different threads). A flowchart of the data analisys performed to merge the incoming data from different sensors is represented in Figure 3.17. The definition of an *already compared couple* condition is able to avoid useless computation.

The results of the performed analisys is published over a set of ROS topic (i.e. one for each user that the system is able to detect) in order to provide the whole architecture with data about the people detected inside the workspace.

[14] `http://openmp.org/`

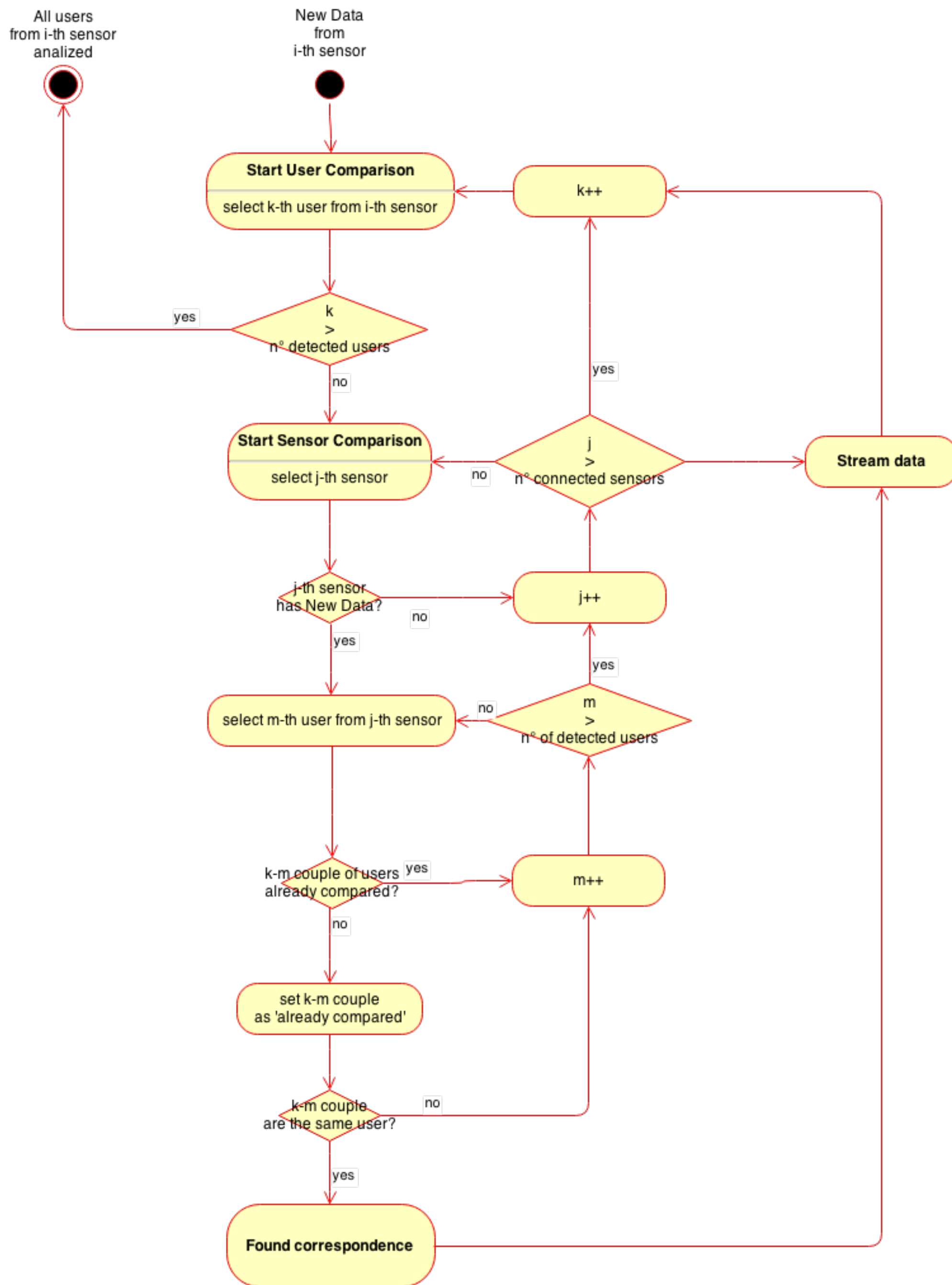


Figure 3.17: Workspace People Detector Flowchart.

3.5 Robot Behavioural control

The final goal of multi-sensor tracking of a robot and of multi-kinect detection of people belonging to the surgical equipe is to perform intra-operative superivision of a robotic procedure, registering possible events that may occur and reacting to them changing the behaviour of the surgical robot. A concrete and robust reaction to a single event should provide the OR with a robotic system able to autonomously adapt the behaviour of its components with respect to the different situations that may happen, ensuring a more flexible human-robot cohabitation inside a sensible workspace (i.e. the OR itself).

In this work we spread the behavioural control of the robot over different components that build up a high-level architecture able to manage different events and situations with respect to a stand-alone and interchangeable *task-descriptor* (i.e. a finite-state machine).

Next we describe the solutions adopted in order to raise events when particular information about either the robot or the workspace are provided by the tracking and people detector agents. First, a description of the scenario of application is provided, togheter with a finite-state machine representation of the specific task that handles information coming from both the multi-sensor tracking agent and the multi-kinect people detector agent. Then, an architecture able to raise events and perform reactions is described, following a *Coordinator-Configurator* pattern able to change the entire architecture topology in reaction to a single event.

3.5.1 Scenario

The case scenario presented in this work is an evolution of the one we presented in [36], which is based on a surgical scenario developed during the European Robotic Surgery (EuRoSurge[15], FP7-ICT-2011-7) project, [37, 38].

A 7 degrees of freedom robot (the presented KUKA LWR4+) performs a target

[15] `http://www.eurosurge.eu/eurosurge/`

approaching task supervised by two different trackers (i.e. the NDI Polaris Vicra and the NDI Optotrack Certus) that perform surgical navigation inside a crowded environment in which the surgical equipe is detected and tracked by a multi-kinect architecture. The preliminary phase of a surgical needle insertion can be a typical case of study for this kind of scenario.

After the surgeon asserts the begin of the procedure, the robot starts moving towards a pre-defined target over the patient skull. If the tracker with the highest priority is able to correctly track the robot inside the OR, the movement is performed with the maximum allowed velocity. In case of fault of the best performant tracker, the second tracker is connected to the architecture and the robot movement is continued. If no tracker is able to correctecly perform navigation, the robot movement is stopped and in order to release the brakes, it is needed to ackowledge the event.

Two different levels of velocity (i.e. *fast* and *slow*) are set for each tracker. The *slow* movement is performed when the robot enters a *critical* area (i.e. a sphere around the patient head). When the robot is outside this area (i.e. *safe* area), the *fast* movement is selected.

People detection allows to have information of the position of the surgical equipe inside the OR. If a high number of users are detected inside an area surrounding the operating table, the *slow* movement condition is triggered and the robot movement slows down. If a possible collision condition is detected (e.g. a user detected too close to the robot during movement) the stop of the robot movement is asserted and the release of the brakes needs to be aknowledged by a user.

3.5.2 Finite-state machine description

In this section a finite-state machine description of the above explained scenario is presented. As already stated (see Section 2.4), finite-state machine provides a simple but powerful way to define the behaviour of a system, togheter with great re-usability, extensibility and modularity of the single behaviour.

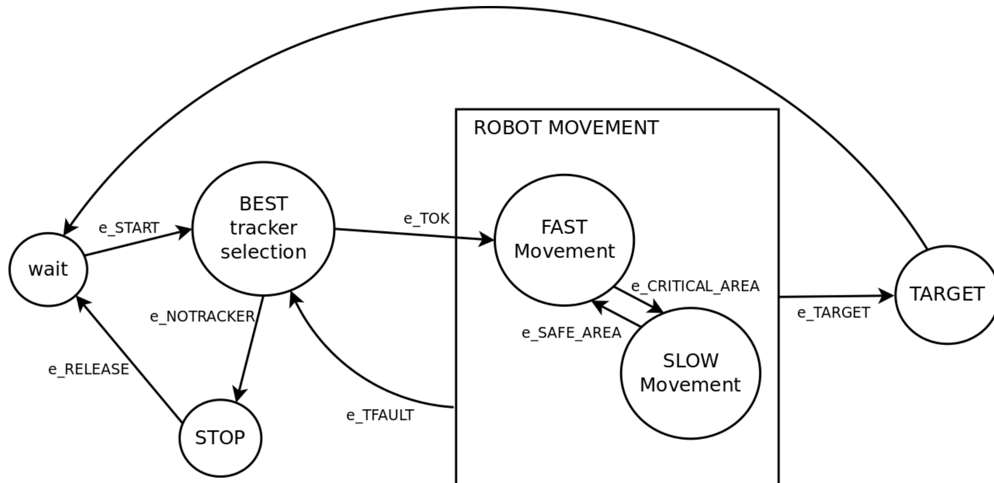


Figure 3.18: Finite-state machine describing the system behaviour. *After a correctly performed startup, the robot enters a `wait` state in which the assertion of the start of the procedure is performed by the surgeon. After the start assertion, the `Best Tracker Selection` procedure is performed in order to retrieve information about the available trackers and to connect the most performant one. If the sensor is found (`e_TOK` event) the robot begins to move. If not (`e_NOTRACKER` event) the robot does not move and brakes are activated instead. Fault of a single tracker causes the selection procedure to be repeated.*

Based on our scenario, each component can be in a certain number of *states*, that fully describe its current behaviour. The transition of each component from one state to another is triggered by events that give to the architecture the complete knowledge of the status of each component. We defined a set of events that describe different situations that may happen in our case study, related to the sensor-side (i.e. the Polaris and the Optotrack system), the surgical equipe (i.e. data incoming from the Microsoft Kinects) and the robot.

A finite-state machine has been designed in order to describe the whole system status and to provide the architecture with information about it. The resulted state machine is represented in Figure 3.18.

After that the start of the procedure is asserted (`e_START` event), the architecture search for the best available sensor able to correctly track the robot movement using the information provided by the Multi-sensor Tracking Architecture. If a sensor is found (`e_TOK` event) the robot begins to move towards the

pre-defined target. The robot performs a *fast movement* outside the critical area (until `e_SAFE_AREA` event) and a *slow movement* otherwise (`e_CRITICAL_AREA`). If no sensor is found during the starting procedure (`e_NOTRACKER` event) the robot movement is not started and the brakes are activated instead. If the connected sensor faults during robot movement (`e_TFAULT` event) the *Best Tracker Selection procedure* is repeated and the new available tracker (if any) is connected. If the robot hits the target (`e_TARGET` event), the movement is stopped and the robot enters a *wait* state. If some conditions has stopped the robot movements activating the brakes, the user's acknowledgment is required to re-start the procedure (`e_RELEASE` event).

3.5.3 Behavioural control architecture

Next we describe the software achitecture used to manage behavioural control of a surgical robot given a finite-state machine of a well-defined task. The UML schema of the developed architecture is represented in Figure 3.19. A **Supervisor** component reads data incoming from the other high-level controllers (i.e. the Multi-sensor tracking architecture and the Workspace People Detector) and the robot position provided by the best available tracker and instructs the robot with the current target and the allowed maximum velocity. The **Coordinator** and **Configurator** components, following what is presented by M. Klotzbücher in [39], act like a stand-alone finite-state machine (i.e. the *Coordinator*) and a separate actions-executor (i.e. the *Configurator*) and are part of a pure Lua module[16].

Supervisor Component. The *Supervisor* is an OROCOS component that acts like an arbiter agent, receiving information and data from all the others architecture modules and raising an event when a particular condition is fullfilled.

The *Supervisor* is deployed as a knowledge-free component with respect to the hardware side of the architecture, meaning that no particular information

[16] `https://bitbucket.org/kmarkus/dng`

Figure 3.19: Behavioural Control Architecture. *The* Supervisor *component reads data incoming from the other high-level controller of the architecture (i.e. the Multi-Sensor Robot Tracker and the Multi-kinect People Detector) and raises events that are sent to the* Coordinator *component. The* Coordinator *is a hierarchical finite-state machine that reads incoming events and request a specific components topology to the* Configurator *component for each transition. The* Configurator *is able to invoke operations and services from all the other components, as well as to create new connections.*

involving the hardware setup are stored into the component at startup and that all the required information are provided run-time from other modules of the system. The only information stored into the *Supervisor* component are task-related and regard the dimension of the *Critical Area* (i.e. the radius of the sphere) and the current target to be reached (i.e. a 3D pose). Those elements are stored into a `.xml` file and loaded during the configuration phase.

Incoming data regarding the *best available tracker* are used to raise events containing information about the tracker itself (e.g. `e_T1`, `e_T2`, etc.) and the fault of a sensor (`e_TFAULT` event).

After the start, the *Supervisor* is connected to the ROS topic of the best available tracker. If the connection succeed, the `e_TOK` event is raised. Once connected, the component continously check the pose of the robot TCP in the intraoperative environment, registering it with respect to to ${}^{Master}RTO$ using the transformation defined in Equation 3.5 and the incoming data (i.e. calibration matrixes and

${}^{LWR}\mathbf{P}$) from the multi-sensor tracking module. If the robot is moving inside the *Critical Area*, that is:

$$\mathbf{d}({}^{LWR,current}\mathbf{P}, {}^{LWR,target}\mathbf{P}) < r \tag{3.11}$$

where $\mathbf{d}(\cdot, \cdot)$ is the euclidean distance between the current pose of the LWR robot (${}^{LWR,current}\mathbf{P}$) and the target pose (${}^{LWR,target}\mathbf{P}$) and r is the Critical Area Radius, the `e_CRITICAL_AREA` event is raised. In case the condition stated in Equation 3.11 is not fullfilled, the `e_SAFE_AREA` event is raised.

Acting the same way, the component checks the pose of each user (which is returned from the Workspace Detector Module) with respect to the LWR robot pose and raises the `e_CRITICAL_AREA` event if too many users are detected near to the robot.

The `e_STOP` event is raised when the Multi-Sensor Tracking Module provides the information that there is no available tracker or when the Workspace Detector Module provides information about a possible collision with a user.

Each event is published over a dedicated ROS topic and sent to the *Coordinator* finite-state machine in order to update the status of the system.

Coordinator Component. The *Coordinator* component is a pure Lua OROCOS component in which is loaded a finite-state machine written using rFSM tool and based on the one presented in Section 3.5.2. This finite-state machine is represented in Figure 3.20 and stored in a stand-alone `.lua` file. During startup, the FSM file is loaded into the *Coordinator* in order to control the system. The component has an `RTT::InputPort` that constantly reads the incoming events from the *Supervisor* component and update the internal finite-state machine. Each transition requires a specific action to the *Configurator* component by sending over an `RTT::OutputPort` a specific component's configuration tag (e.g. `tracker_1_up`). The *Configurator* will apply the requested configuration and return an acknowledge tag, allowing the *Coordinator* to enter the target-state of the current transi-

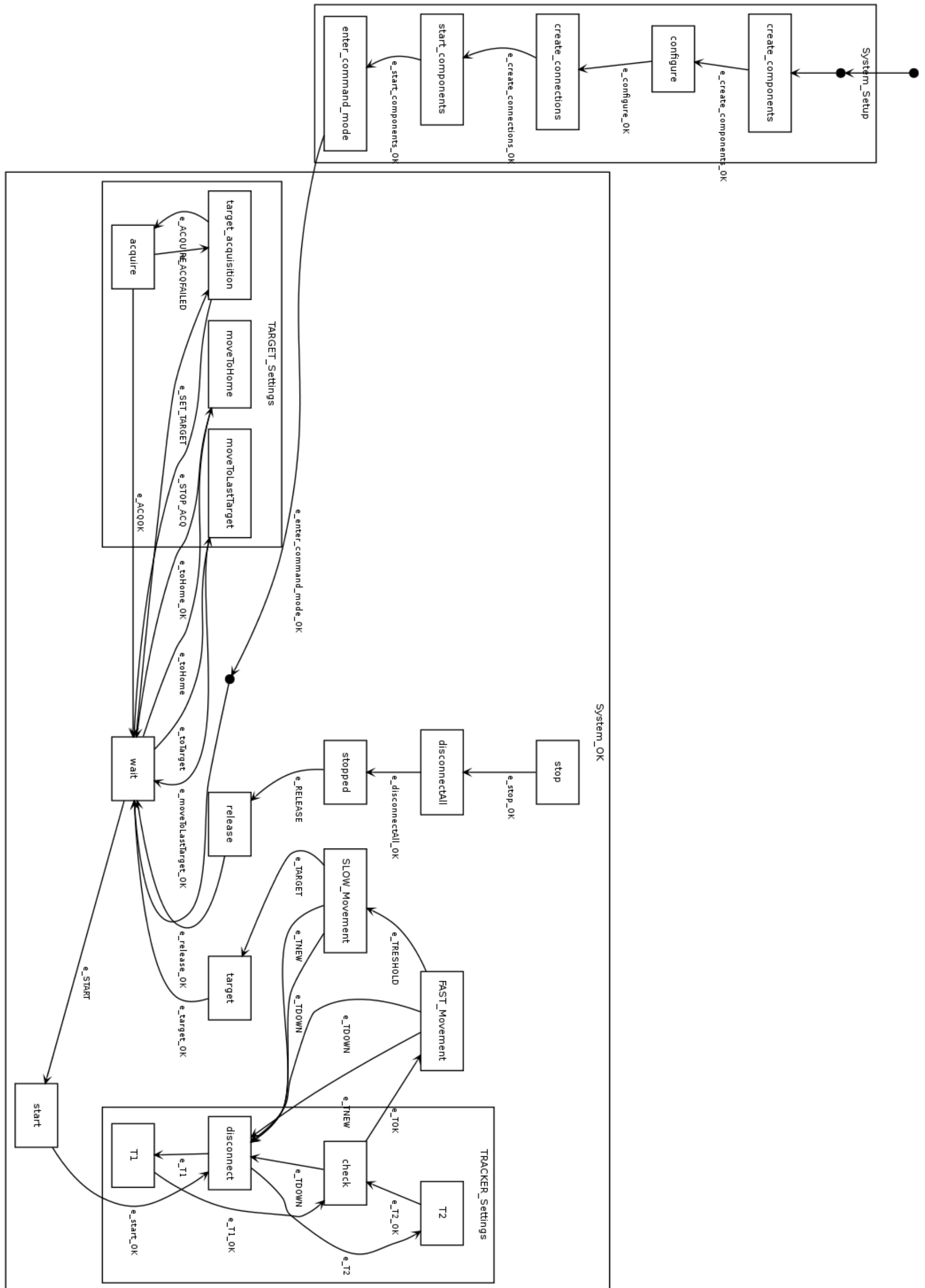


Figure 3.20: Developed finite-state machine for robot behavioural control. *The represented FSM is written using rFSM tool and loaded at startup inside the Coordinator component. The two top-level state machines handle both the system setup and configuration (System_Setup* state machine) and the task execution (System_OK *state machine). Failure during the setup cause the architecture to abort the process. This is an automatic representation generated using rfsm2uml tool.*

tion.

A preliminary (linear) finite-state machine (i.e. *System_Setup*) that requests to the *Configurator* a set of actions in order to setup the whole architecture is developed and ensures automatic preparation and configuration of all the involved components. If the setup is accomplished, the *Coordinator* enters the main finite-state machine (i.e. *System_OK*) that represents the real task-descriptor. If some components startup or configuration fail, the setup is aborted and the robot movement cannot be started.

Configurator Component. The *Configurator* component is a pure Lua OROCOS component that acts like an action-executor in response to a specific request. The component is directly connected to the *Deployer* component and, using the OROCOS `RTT::OperationCaller`s is able to invoke a set of atomic actions, i.e.

- to `load` a package;
- to `create` a component;
- to `set` a property from a `.cpf` file;
- to `connect`, `read` or `write` a Port;
- to `call` an operation from a specific component.

Combining these simple operations, it is possible to create a `.lua` file containing a set of pre-defined components configurations (i.e. a specific topology and the status of each component) that can be requested at any time during the system execution by the *Coordinator*. An example of configuration (i.e. `tracker_1_up`) is reported below:

```
T1 = config {
   pre_conf_state = { "supervisor:Stopped", "default:Ignore" },
   post_conf_state = { "supervisor:Running", "default:Ignore" },
```

```
    call{ op="supervisor.LWR_pose.disconnect" },
    call{ op="supervisor.Reference_pose.disconnect"},
    connect{ from="supervisor.LWR_pose", to="LWRCertus",
             connpol={  transport=3,
             size=1,
             name_id="/certus/calibration_tool_optical" }
    },
    connect{ from="supervisor.Reference_pose", to="RefCertus",
             connpol={  transport=3,
             size=1,
             name_id="/certus/reference_tool_optical" }
    },
    call{ op="supervisor.changeMovementParameters" },
}
```

First, the *Supervisor* component is stopped (other components, i.e. `default`, are left at their current status, i.e. `Ignore`). Then the current tracker is disconnected and the `T1` tracker is connected to the respective `InputPorts`. Finally, movement parameters are modified accordingly with the new connected tracker (calling an `Operation` of the *Supervisor* component) and the *Supervisor* is switched to `Running` status again.

3.6 Application

In this section we apply the developed high-level control architecture to a real robot (i.e. the KUKA LWR4+ described in Section 3.1.1). First we provide a description of the complete architecture deployed, that uses the three described high-level controller object of this work togheter with a *goTo* module that takes a target and generates a trapezoidal-velocity trajectory, sending it to the robot controller and managing velocity changes and eventual robot brakes. Next we describe a Graphical User Interface (GUI) in order to provide the final user (i.e. the surgeon) with a friendly interface and a visual feedback of the whole system status. Finally, we describe a set of tests that we performed in order to evaluate the system response and ability to control the robot during emergency situations.

3.6.1 Complete architecture

The complete architecture used to perform a robot target approaching task during, for example, the preliminary phase of a needle insertion in the surgical scenario (described in Section 3.5.1) is composed by five agents:

- the *Multi-sensor Tracking* module, described in Section 3.3.4;
- the *Multi-kinect Workspace People Detector* module, described in Section 3.4.4;
- the *Behavioural Controller* module, described in Section 3.5.3;
- a *goTo* module that reads the incoming provided target and velocity parameters and generates a trapezoidal-velocity trajectory;
- a *Robot Interface* module (i.e. the KUKA Fast Research Interface, FRI[17]).

The complete architecture is represented in Figure 3.21, in which all the involved components (software and hardware) are shown with their respective connections

[17] http://git.mech.kuleuven.be/robotics/kuka_robot_hardware.git

and exchanged data. Next we provide a description of the *goTo* and *FRI* modules used in this work. The former is composed by a **Cartesian Interpolator** component[18] that handles the target and the current position of the robot, providing the above described trajectory in the cartesian space. The latter is composed by the **FRI** component itself, that is used coupled with a **Robot Impedance Controller** (that manages stiffness and damping of the LWR4+ robot) and with a **FRI Master** component (that allows us to automatically switch from a control modality to another).

Cartesian Interpolator Component. The *Cartesian Interpolator* component is an OROCOS component that interpolates a set of variables (i.e. the components of a position vector and of a quaternion) between an initial value to a target value with a trapezoidal-velocity trajectory.

At startup, the component loads a set of configurations from a `.cpf` file, regarding the maximum allowed velocity and acceleration to be used during interpolation. The maximum allowed velocity is the velocity used in the constant-velocity phase of the trapezoidal trajectory. If the maximum velocity constraint is changed run-time during the interpolation, the component performs a velocity ramp with constant acceleration to hit the new velocity constraint. If the stop of movement is requested, the component performs a trajectory that leads to bring the commanded system (in our case, the robot) to the position where the stop was asserted.

Fast Research Interface Component. The Fast Research Interface is a high-level communication protocol developed by KUKA Roboter to allow external control of the LWR4+ robot, [41]. The robot KRC controller and the external host are connected through an ethernet cable and exchange data with an update rate up to 1 KHz using a standard UDP datagram. The modality of control allowed are 3, i.e. a Joint Position Control, a Joint Impedance Control and a Cartesian

[18] Contributions from M.D. Comparetti, *Ph.D.*, [40]

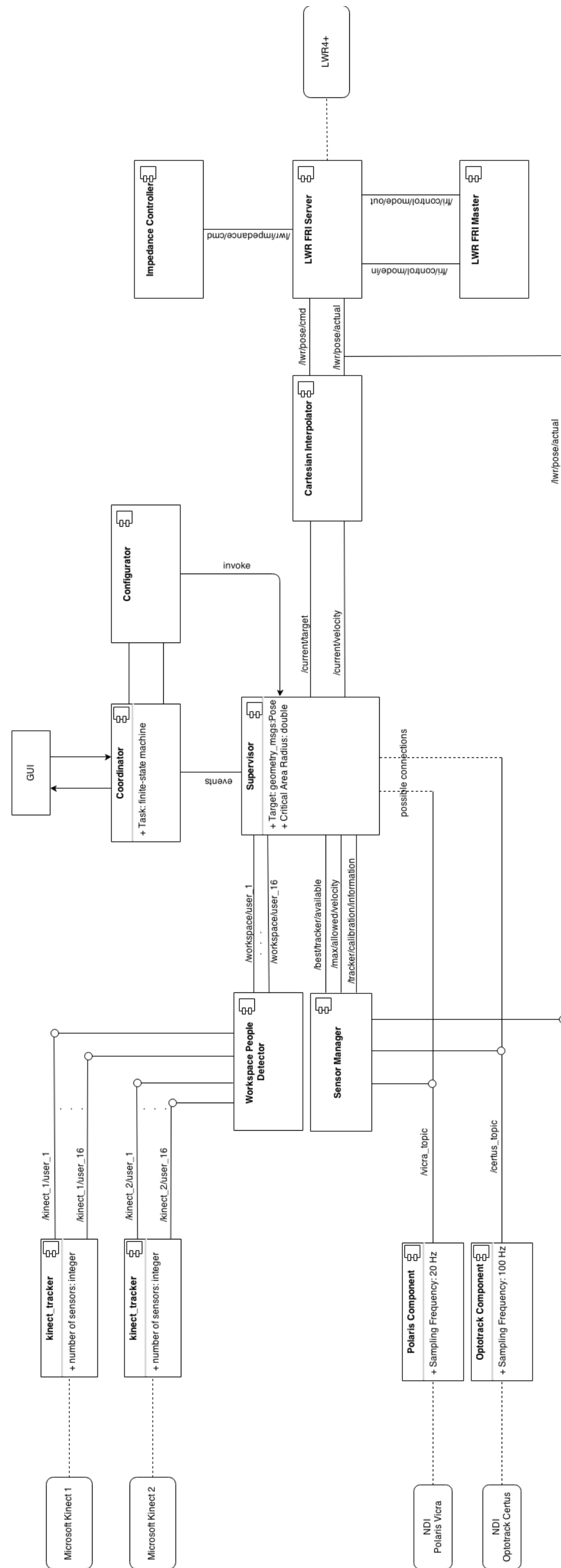


Figure 3.21: Complete control architecture.

Admittance Control, all developed by DLR[19] (Germany, Oberpfaffenhofen) and included in the low-level KUKA LWR4+ controls.

The *FRI Component* is an open-source OROCOS component developed from the K.U. Leuven group (Leuven, Germany, [20]) in order to manage the communication with the LWR4+ in OROCOS environment and through the FRI communication protocol.

3.6.2 Graphical User Interface

The main problem about ROS and OROCOS robot architectures when addressing the final user is the low user-friendly approach of deployment and control of the components, which is normally achieved by `.xml` schema of components topology and command-line actions. Different solutions have been provided, in particular regarding the Graphical Interface approach to the control of a system. A common used tool is `rqt`[21], a Qt-based framework for graphical user interfaces development inside ROS environment. Qt[22] are widely known libraries for UI design that support different programming languages (e.g. C++, phyton, etc.).

During this work, we developed a Graphical User Interface (GUI) that works inside ROS as a rqt plugin and allows the user to send events to the *Coordinator* finite-state machine and to have a visual feedback of the whole system status. The resulted GUI is represented in Figure 3.22.

The `START` button on the left side allows the user to assert the begin of the procedure. On the right, a system status overview is reported and information of the operation currently performed by the robot (i.e. `wait`, `approaching target`, `stopped`) are provided. Information about the area of movement are provided in the status bar following, together with knowledge of the used tracker (i.e. `1`, `2`, etc. or `No Sensor Available`). The `Go To:` section allows the user to choose

[19] `http://www.dlr.de/`
[20] `http://www.kuleuven.be/`
[21] `http://wiki.ros.org/rqt`
[22] `http://qt-project.org/`

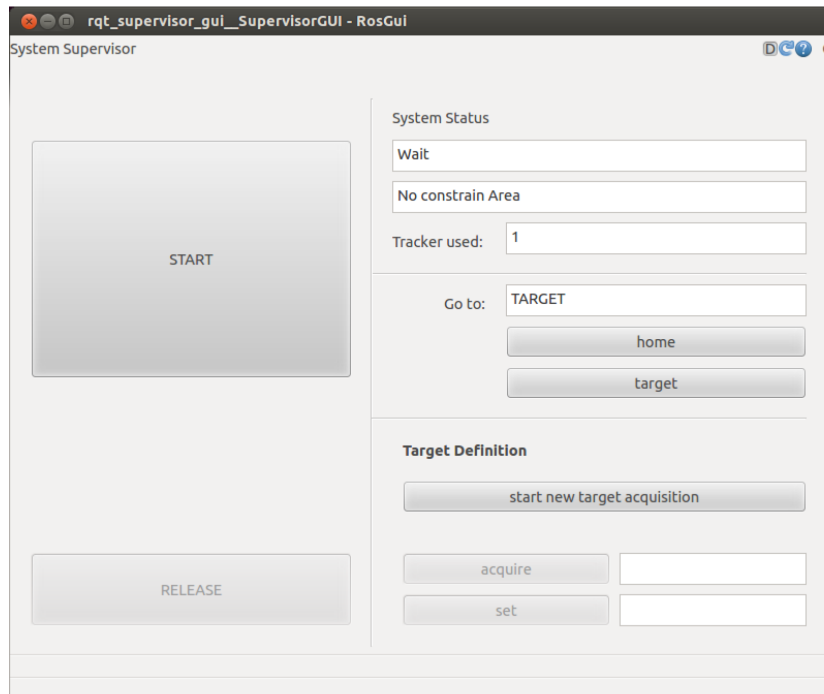
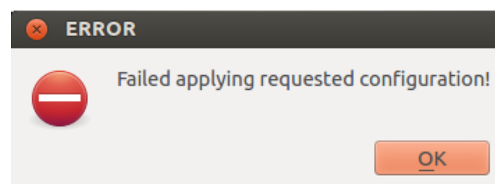


Figure 3.22: Graphical User Interface.

the current target of the procedure between the pre-defined target on the patient head or a standard home position. If during the procedure something happens that leads the robot to activate the brakes, the `Release` button allows the user to acknowledge the system failure, release the robot brakes and re-start the procedure. Finally, the `Target Definition` section is an utility that allows to define a specific target run-time, without the need to use a pre-defined target stored in a `.xml` file.

The developed GUI is able to provide the user with a feedback also if some of the components setup or configuration faults using a standard Qt error window, thus



The acknowledgement of the fault causes the application to close.

3.6.3 Performance tests

In this section we describe the experimental protocol used to test the architecture performances when responding to different emergency situations. We collect latencies of response to a specific event, undesired displacement of the robot during the braking ramp and we perform a statistical analysis of the performance of the multi-kinect approach in people detection.

A. Multi-tracker robot navigation tests

Next, a description of the tests performed in order to evaluate the multi-tracker navigation architecture is provided. The performed tests aim to evaluate the whole system's response after emergency situation like the fault of a single tracker or the unavailability of sensor. A test to evaluate the architecture repeatability of response performances is also presented.

- **Sensor Failure Reaction Test** (*Sensor Swap*). In this test we simulated the fault of a single tracker by occluding the FOV, in such a way that it is no more able to provide the position of the robot. The latency from the fault recognition and the connection of the best available tracker is measured using ROS Time Stamps recorded with a specifically designed component deployed together with the whole architecture and commanded by the *Configurator*. The experimental protocol is the following:

  - At the beginning, the two considered trackers are active and correctly tracking the robot;
  - After the `START` assertion, the first tracker is connected to the architecture;
  - The FOV of the tracker is occluded. Time Stamp of the failure recognition ($t_{trackerfail}$) is acquired;

- The first tracker is disconnected and the second tracker is connected. Time Stamp of the first valid data incoming from the new tracker ($t_{newtracker}$) is acquired;
- Latency is computed as:

$$L^{SensorSwap} = t_{newtracker} - t_{trackerfail} \quad (3.12)$$

The test is performed 50 times and median and interquartile range are computed.

- **No Available Sensor Reaction Test** (*No Sensor To Stop*). In this test we simulated the contemporary fault of all trackers by occluding their FOV. The latency between the recognition of the unavailability of sensors and the `STOP` command sent to the robot is measured using a specifically designed component. The experimental protocol is the following:

  - At the beginning, the two considered trackers are active, but only one is able to correctly track the robot movement;
  - After the `START` assertion, the only available tracker is connected to the architecture;
  - The FOV of the tracker is occluded. Time Stamp of the failure recognition ($t_{trackerfail}$) is acquired;
  - The tracker is disconnected and the architecture search for a new tracker to connect;
  - Since no tracker are available, an emergency is raised and a `STOP` command is sent to the robot. Time Stamp of the emergency recognition ($t_{nosensors}$) is acquired;
  - Latency is computed as:

$$L^{NoSensorToStop} = t_{nosensors} - t_{trackerfail} \quad (3.13)$$

The test is performed 50 times and median and interquartile range are computed.

- **Repeated Sensor Failure Reaction Test**. In this test we repeatedly simulate the failure of a single sensor by occluding its FOV in order to test the repeatability of the response performance. First, we simulate the fault of a single tracker by occluding its FOV. The latency of response between the failure recognition and the connection of the new available tracker is measured. Then, the tracker FOV is uncovered (in order to trigger the architecture to reconnect to it) and then occluded again. The occlusion operation is repeated up to 50 times, measuring the latencies with ROS Time Stamps. The experimental protocol is the following:

    - At the beginning, the two considered trackers are active and correctly tracking the robot;
    - After the `START` assertion, the first tracker is connected to the architecture;
    - The FOV of the tracker is occluded. Time Stamp of the failure recognition ($t_{trackerfail}$) is acquired;
    - The first tracker is disconnected and the second tracker is connected. Time Stamp of the first valid data incoming from the new tracker ($t_{newtracker}$) is acquired;
    - Latency is computed as:

    $$L_i^{SensorSwap} = t_{newtracker} - t_{trackerfail} \tag{3.14}$$

    where i represents the repetition index.

    - The complete operation is repeated for $i = 1, \cdot, 50$ and latencies are acquired every time.

The whole procedure is repeated 30 times and median and interquartile range of the corresponding latencies are computed.

B. Multi-kinect people detection tests

The tests performed in order to evaluate the performace of the multi-sensor algorithm are described in this section. Performance regarding the capability to discriminate between different users in the same workspace as well as to recognize that different users are actually the same user are analyzied. Next, latencies of response to a possible user collision are measured.

- **People Detection Test**. In this test we measure the performance of the people detection algorithm when recognizing a well-known number of users in the workspace. A growing number of users are asked to enter and exit the workspace and the number of users detected by the architecture is measured. A statistical analysis is carried out, in which the algorithm returns a positive value (P) if there a user is detected and a negative value (N) elsewhere. We defined (1) a true positive (TP) if a user is recognized inside the workspace *and* the user is actually present, (2) a false positive (FP) if a user is recognized inside the workspace when he is not present, (3) a false negative (FN) if the user is present inside the workspace and it is not recognized by the architecture and a (4) true negative (TN) if the user is not recognized and he is not inside the workspace. Using the acquired data, we computed accuracy (ACC), sensitivity (or true positive rate, TPR) and specificity (or true negative rate, TNR) of the developed algorithm as follows:

$$ACC = \frac{TP + TN}{P + N} \tag{3.15}$$

$$TPR = \frac{TP}{P} \tag{3.16}$$

$$TNR = \frac{TN}{N} \tag{3.17}$$

Moreover, in order to have information about the probability of success when detecting a user, we performed an exact binomial test in which the success is represented by the algorithm correctly recognizing (or not recognizing) a user (i.e. $success = TP + TN$) and the failure is represented by the other cases (i.e. $failure = FP+FN$) with the null hypotesis that our distribution belongs to a binomial ($p = 0.8$) and using a confidence interval of 0.95.

- **Collision Reaction Test**. In this test we simulate an emergency collision event by asking a correctly-tracked user to touch the robot during its movement. The latency of response between the recognition of the possible collision and the `STOP` command sent to the robot is measured using ROS TimeStamps. The experimental protocol is the following:

  - A user is asked to enter the robot workspace and to be recognized by the People Detection module;
  - After the recognition, the `START` of the robot movement is asserted;
  - The user is asked to touch the robot to trigger a collision emergency. Time Stamp of the collision recognition ($t_{collision}$) is acquired;
  - The collision triggers an emergency `STOP`. Time Stamp of the `STOP` command sent to the robot (t_{stop}) is acquired;
  - Latency is computed as:

$$L^{CollisionReaction} = t_{stop} - t_{collision} \tag{3.18}$$

  The test is repeated 50 times and median and interquartile range are computed.

C. Robot behavioural control test

Here we describe a test performed in order to evaluate the ability and the performance of the robot to change its behaviour in response to a set of emergency situations.

- **Emergency Brake Test**. In this test we simulate an emergency condition that triggers a `STOP` command sent to the robot in order to evaluate the braking performance. The robot movement is started with a planar trajectory (i.e. a movement only along the (x, y) plane). The `STOP` assertion is triggered (e.g. by occluding all trackers FOV) and the displacement of the robot TCP from the command assertion and the actual stop of the movement is measured with a specifically designed component commanded by the *Configurator*. The test is repeated for different values of velocity of movement (i.e. 1, 2.5, 5, 7.5, 10 cm/s) and root mean square error is computed in order to evaluate the performance.

4 | Results

4.1 Overview
4.2 Multi-sensor robot tracking
4.3 Multi-Kinect people detection
4.4 Robot behavioural control

4.1 Overview

In this chapter results of the tests described in Section 3.6.3, in which we apply the developed architecture to the scenario explained in Section 3.5.1, are reported. Data regarding latencies are acquired using Time Stamps from the CPU in two different moments. Data regarding displacements are acquired using the 3D pose of the robot TCP returned from the KUKA FRI or from the navigation system.

4.2 Multi-sensor robot tracking

Regarding the run-time switch between two different sensors, an example of the detected pose of the tracked robot when changing from one sensor to another is shown in Figure 4.2. Results show a small difference between the robot TCP pose detected from one sensor and the same pose detected by the other sensor. Latencies of response of the architecture after the recognition of a single sensor fault (*Sensor Swap* test) or after the acknowledgment that no sensor is correctly

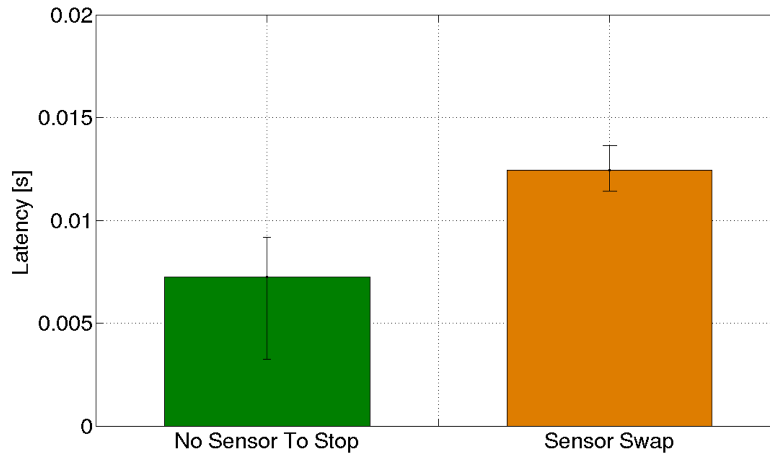


Figure 4.1: Sensor fault reaction latencies. *Results are reported in terms of mean and standard deviation. The* No Sensor To Stop *results in a better performance (a median of 7.3 ms) while the* Sensor Swap *has worse performance (a median of 12.5 ms) because of the major number of computation required. In terms of interquartile range, the two tests perform differently, with values of 5.9 and 2.2 ms respectevely.*

providing the pose of the surgical robot (*No Sensor To Stop* test) are reported in Figure 4.1. The *No Sensor To Stop* test results to perform better (a median of 7.3 ms). The *Sensor swap* test shows a worse performance (a median of 12.5 ms). In terms of interquartile range, the two test performs differently (5.9 ms and 2.2 ms respectevely).

Regarding the time dependancy and the repeatability of architecture's response to a repeated failure of a single tracker, represented as a boxplot in Figure 4.3, median and interquartile range (IQR) are reported. The worst performance in terms of median is at the 47th repetition, that shows a median of 14.6 ms, while the best performance (a part for the first repetition) in terms of median occurs at the 17th repetition, with a registered value of 11.1 ms. The first repetition refers to the first connection (without the disconnection) of the best available tracker and thus it shows a median of 7.39 ms because of the fewer number of computation required. A wider variability can be observed in the interquartile range, with a maximum range of 10.55 ms registered at the 26th repetition and a minimum range of 1.45 ms registered at the 2nd repetition.

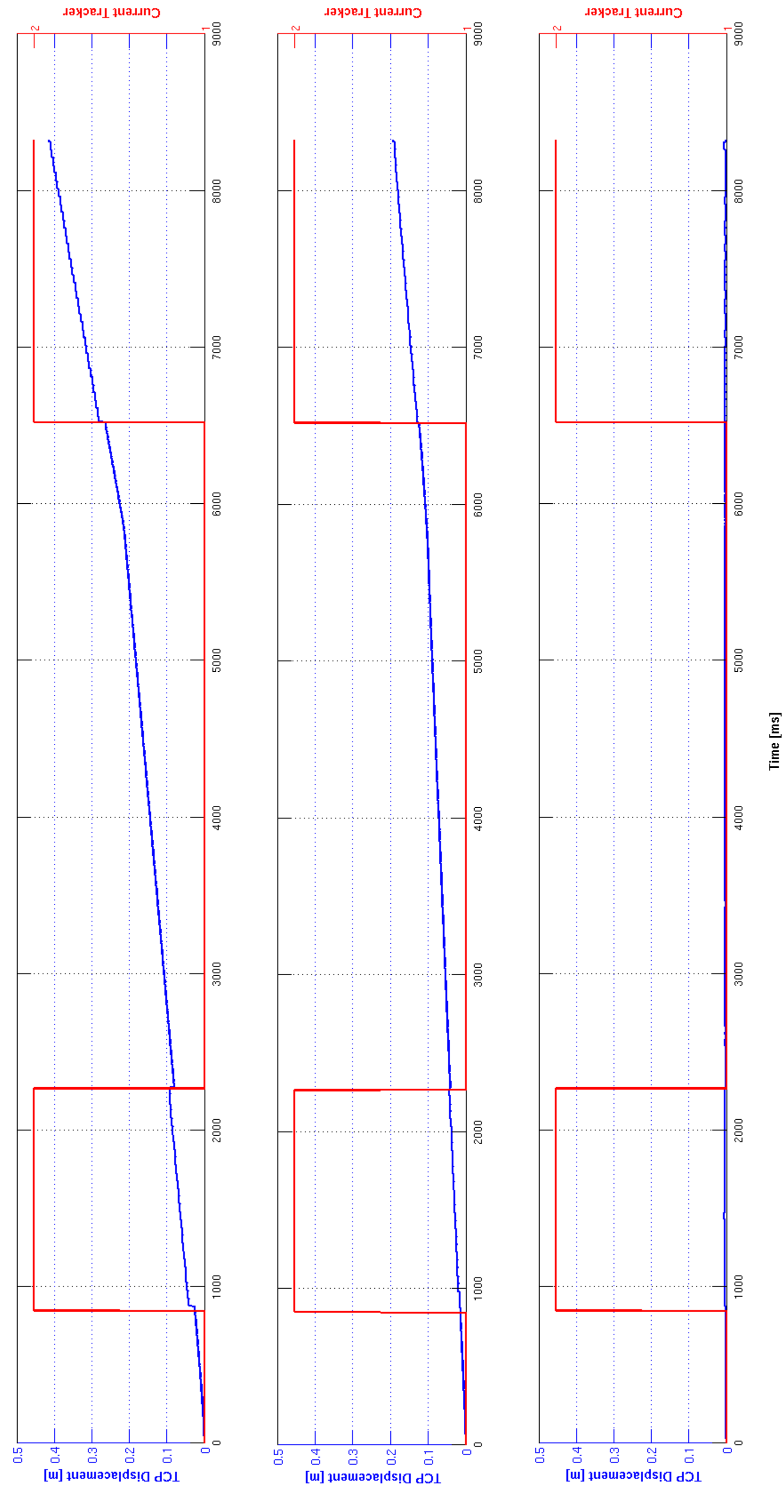


Figure 4.2: Example of sensor swap during robot movement. *TCP trajectory for the robot moving at 7.5 cm/s is shown over the three axis with the blue line. Sensors swap are underlined with the red line.*

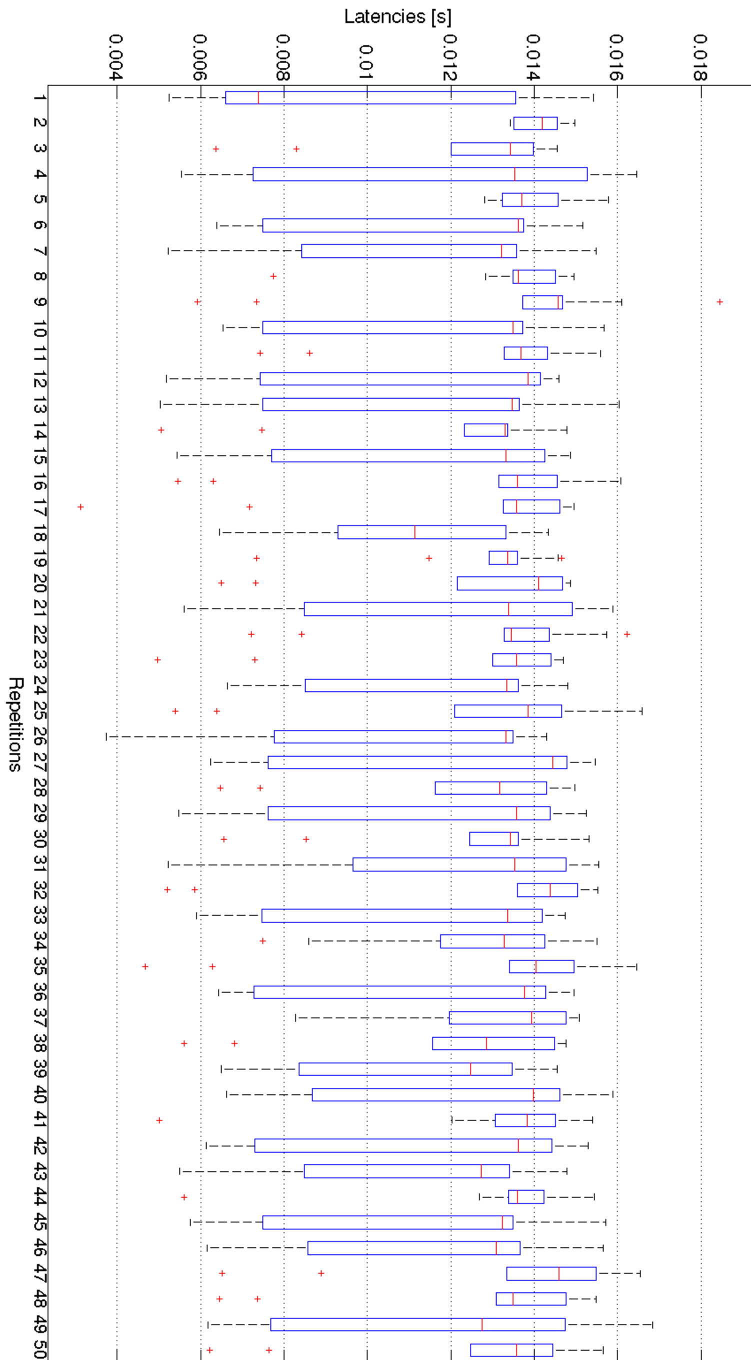


Figure 4.3: Repeated sensor fault reaction latencies. *Results are shown in terms of median and interquartile range. Red stars are related to outliers.*

4.3 Multi-Kinect people detection

Results of the performed calibration between paired Kinect are shown in Figure 4.4. Residuals show a median error of 35 mm and a wide interquartile range (more than 20 mm).

Results regarding the ability of the Workspace People Detection Module to recognize a user inside the workspace of the Kinects are reported in Table 4.1 for tests performed with one and two users inside the workspace respectevely. In both tests, the algorithm's results showed to correctly detect and track the people inside the environment. In a limited number of cases, 5 and 1 respectevely for the test with 1 and 2 users, a false positive (thus, 1 user more than the correct number detected inside the workspace) appears for a limited amount of time (less than 1 s). This condition is indicated with (*) in Table.

Users			
1		*Positive*	*Negative*
	True	30	30
	False	0 (5*)	0
2		*Positive*	*Negative*
	True	60 (30×2)	60 (30×2)
	False	0 (1*)	0

Table 4.1: Multi-Kinect people detection performance. *The algorithm performed succesfully (i.e. detected the correct number of users inside the workspace) at each test. In a limited number of cases (indicated with (*) in Table) a false positive (i.e. a user more than the actual number) appears for a limited amount of time (less than 1 s).*

Considering this worst case as the gold standard, we computed accuracy, sensitivity and specificity. Resulted values are reported in the Table 4.2.

Users	*Accuracy*	*Sensitivity*	*Specificity*
1	0.92308	0.85714	1.0
2	0.99171	0.98361	1.0

Table 4.2: Multi-Kinect people detection accuracy, sensitivity and specificity.

Moreover, the performed test returned that the values distribution belongs to

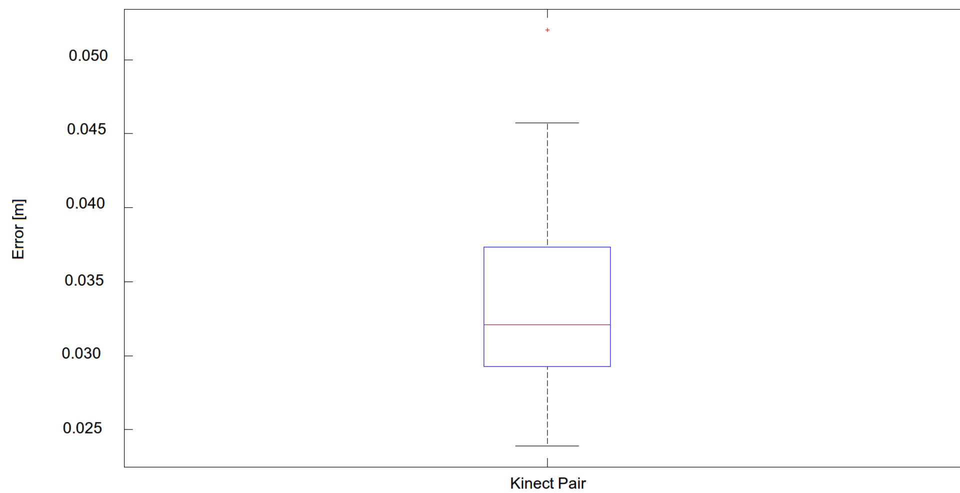


Figure 4.4: Calibration error for the considered couple of Kinect Sensor. *Results are shown in terms of median and interquartile range.*

a binomial with $p = 0.8$ ($p-value < 0.05$) and computed an estimated probability of success from the sample of 0.9167.

Regarding latencies of the architecture reaction when detecting a possible collision (Figure 4.5), results show a median value of 1.1 ms with an interquartile range of 0.3 ms.

Figure 4.5: Collision reaction latencies. *Results are shown in terms of median and interquartile range.*

4.4 Robot behavioural control

Regarding the behavioural control and thus the ability of the developed supervision system to change the behaviour of the surgical robot in reaction to an incoming event, examples of sent commands and robot TCP responses are reported in Figure 4.6 and Figure 4.7.

In Figure 4.7 an example of the robot control during an emergency situation that triggers a `STOP` command is shown. Experimental condition are described in Paragraph 3.6.3.C togheter with the tests performed. After the `STOP` is asserted, the Supervision System generates a velocity trajectory in order to brake the robot and to bring the TCP back to the position where the emergency was received. The position command shows an almost null final displacement ($displacement < 0.5mm$) on all the three axis. The actual position of the robot TCP shows a higher displacement ($displacement > 5mm$), with the maximum value located on the x axis. Results are shown in terms of median (solid lines) and interquartile range (shaded areas) over 60 samples.

In Figure 4.8 results regarding the robot TCP displacement after a `STOP` assertion when moving with different cartesian velocities (i.e. 1.0, 2.5, 5.0, 7.5, 10.0 cm/s) are shown in terms of Root Mean Square Error (RMSE) with respect to the commanded position. The best performance is registered when moving at the minimum considered velocity (i.e. 1.0 cm/s) with a median RMSE of 0.64 mm, 0.36 mm and 0.029 mm respecevely for the x, y and z axis. Interquartile range at this velocity shows a maximum value of 0.11 mm registered for the y axis. The worst performace is registered at the maximum velocity of 10 cm/s, with a median RMSE of 5.9 mm, 3.4 mm and 0.19 mm respecevely for the x, y and z axis. Interquartile range at this velocity shows a maximum value of 0.78 mm registered for the y axis.

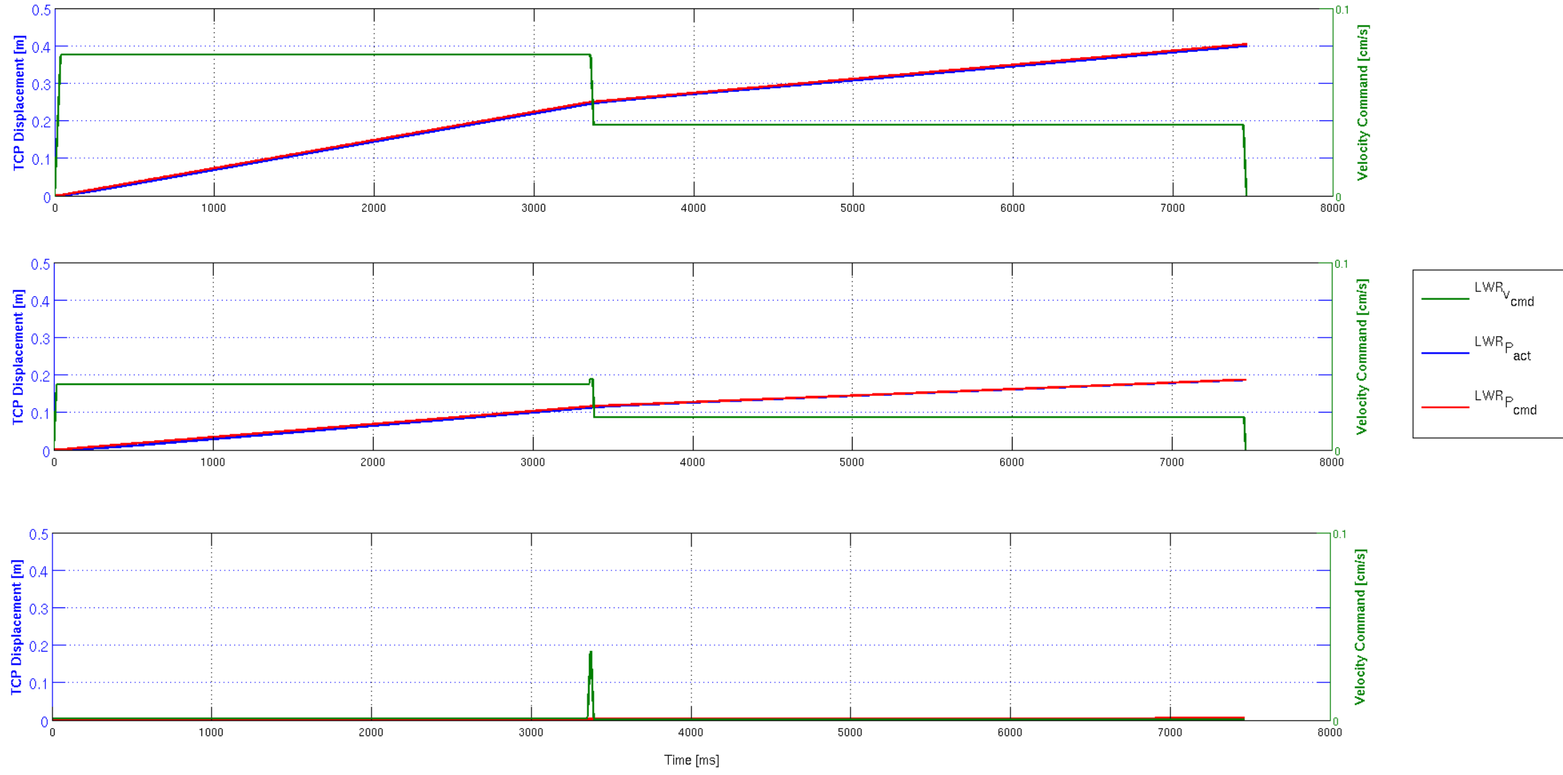


Figure 4.6: Example of the trajectory commanded to the robot. *Commanded velocity and position are shown togheter with actual position of the TCP for the three axis. Red lines refer to the commanded position, while blue lines refer to the actual position. Green lines refer to the velocity command. In this example, the robot is moving with a velocity of 7.5* cm/s.

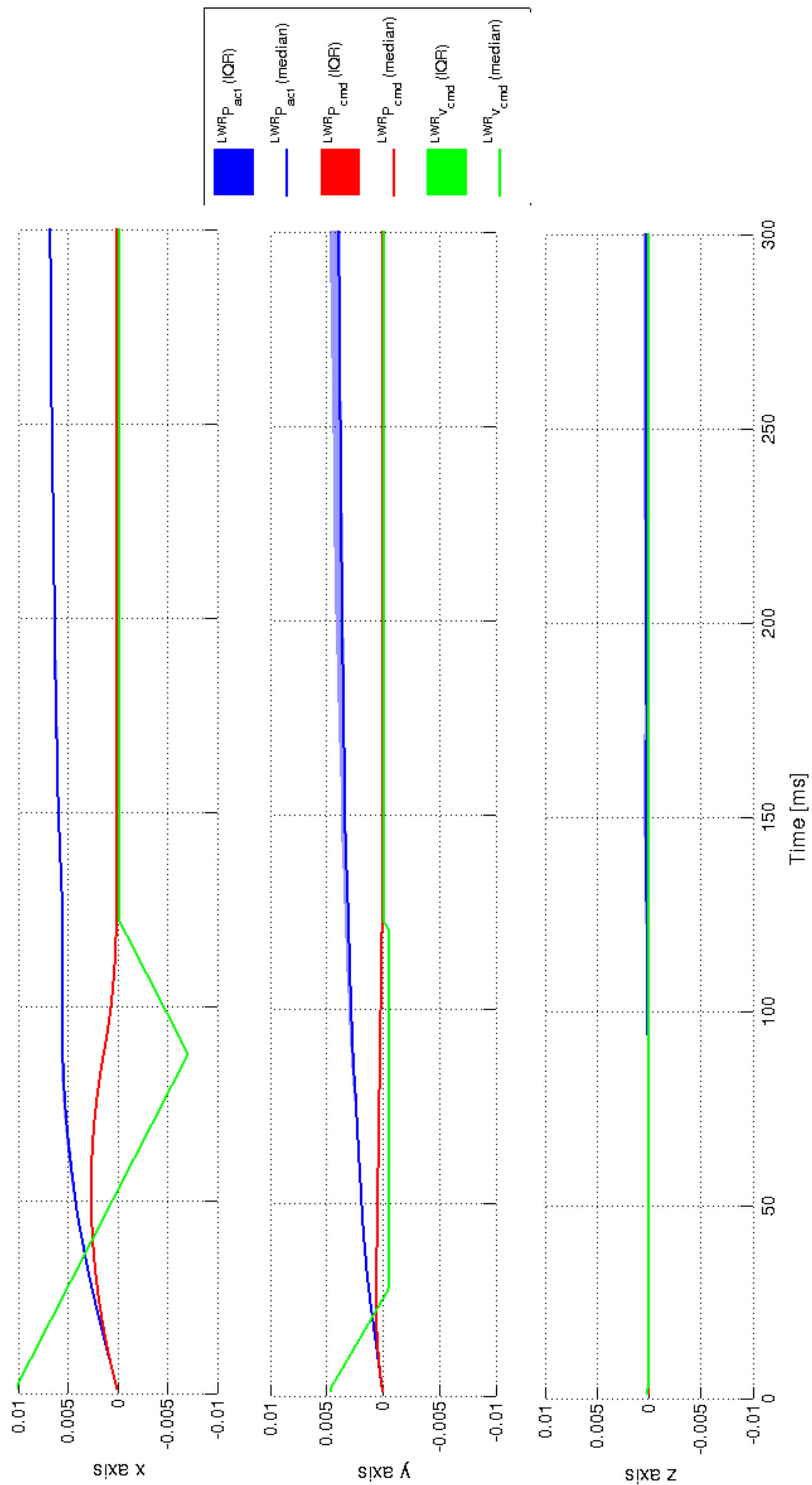


Figure 4.7: Example of emergency STOP command and TCP displacement. *In this example, the robot is moving with a cartesian velocity of 10 cm/s. Results are shown in terms of median (solid lines) and interquartile range (shaded areas).*

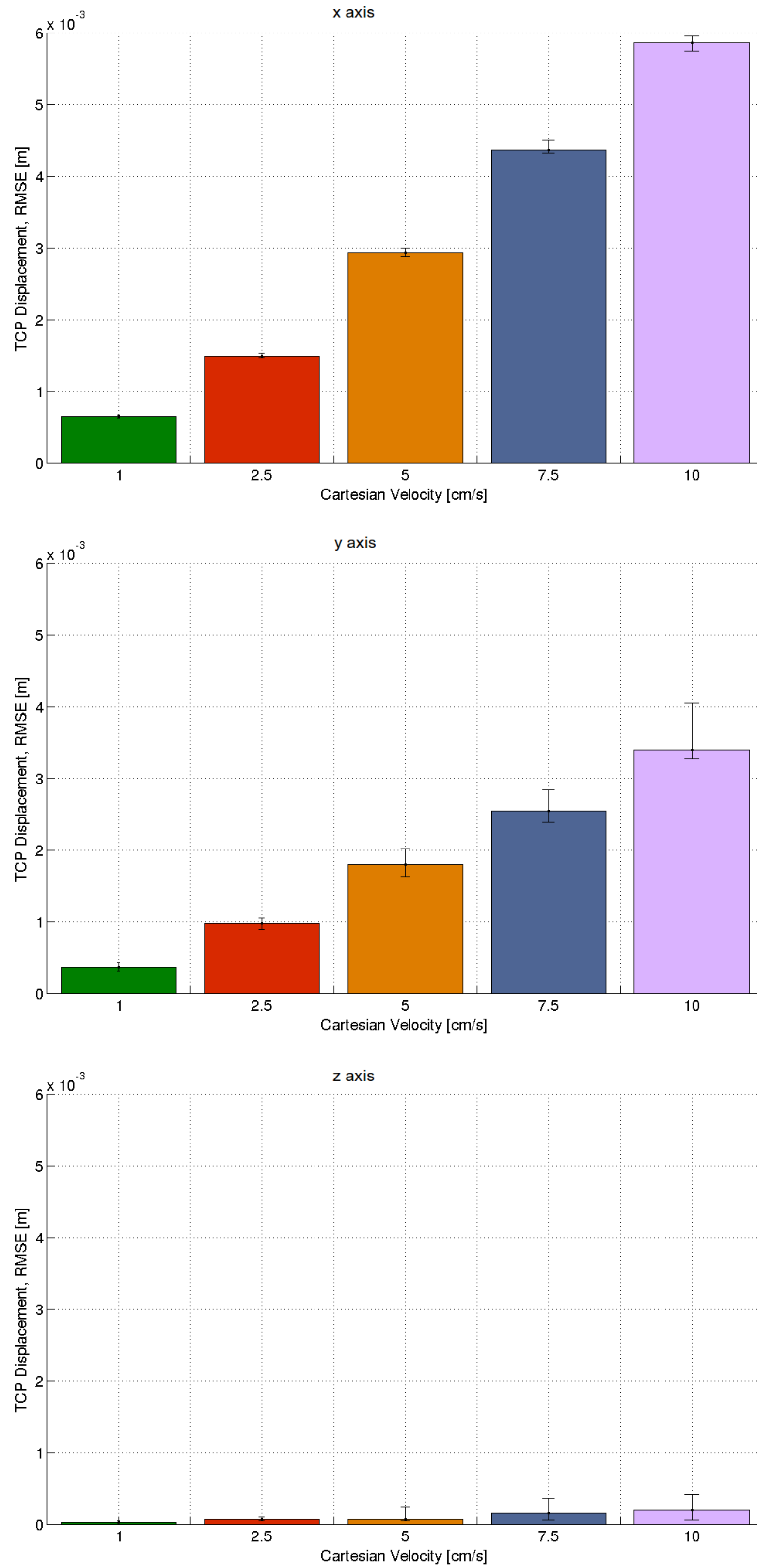


Figure 4.8: RMSE of the TCP Displacement after `STOP` assertion. *Results are shown in terms of median and interquartile range.*

5 | Discussions and conclusions

In last few years, research in the field of surgical robotics is more and more evolving in a direction that aims to provide a versatile and intelligent assistance system able to autonomously take care of the human-robot interaction inside a critical environment like the OR, handling emergencies that a shared environment may arise. Such a system should be developed so that people inside the OR do not have to take care of the robot itself, thus guaranteeing a more comfortable cohabitation (i.e. with the surgeons free to move around the robot even during its movement) but maintaining safety and standard features of robotics in surgical applications. For example, intraoperative tracking is a primary need in surgical robotics, where intrinsic accuracy of involved systems does not respect the requirements of such a critical application [42]. Targeting algorithms to solve this issue have been developed (e.g. the one presented in [43]) and shows a sub-millimetric accuracy but suffer from sensors occlusion inside crowded environments. Moreover, if the surgical equipe is free to move around the robot during its movement, a robust human detection and collision avoidance must be provided in order to ensure safety.

In this work we presented a high-level software architecture that aims to supervise a surgical semi-autonomous robotic system in a crowded environment like the OR, ensuring robust intraoperative tracking with a multi-sensor approach and detection and tracking of the surgical equipe using RGB-D cameras. Reactions to

the particular emergencies that may occur are performed updating the behaviour of the robot with respect to the situation detected.

The developed architecture is completely modular, scalable and configurable. All the needed parameters are loaded at startup from `.xml` files and thus can be easily retrieved from database or provided by shared knowledge about the procedure and the involved components (e.g. from an ontology). Moreover, the information about the task to be performed are stored in a single file (i.e. a `.lua` finite-state machine) loaded during the setup of the application into the *Coordinator* component and thus can be hardly changed by the modification of this single file.

Hardware independency is guaranteed for all the components, with exception of the modules that directly take care of the communication with the involved hardware. Since the entire approach followed is characterized by the redundancy of sensors, there is no teorical limit to the number of sensors (i.e. trackers or RGB-D cameras) that can be connected to the architecture without the need to recompile. Nevertheless, the architecture is correctly working also without sensor's redundancy (i.e. with only one tracker and one RGB-D camera).

Regarding intraoperative tracking, results showed that is possible to run-time switch from one sensor to another in order to always use only the best performant tracker currently available with only a small oscillation of the detected pose. This oscillation should be related to the calibration residuals that differs between the two sensor (Table 3.4). Latencies of the swap operation resulted in a wide variability but with a limited maximum value (Figure 4.1 and 4.3 in Chapter 4). In the case of the maximum cartesian velocity considered in this work (i.e. 10 cm/s) this leads to less than 2 mm of untracked movement before the connection of the new tracker. The choice of a cartesian velocity of 7.5 cm/s ensures reaction to a sensor fault with sub-millimentric accuracy. The recognition that no sensors are able to correctly track the robot inside the surgical workspace performed better and guar-

antees a reaction (i.e. a `STOP` of the movement assertion) whithin sub-millimetric movement at every considered velocity.

Regarding human detection inside a critical environment like the OR, results (Table 4.1 in Chapter 4) showed a good ability in detecting and tracking a user inside the workspace of the chosen RGB-D cameras (i.e. a set of Microsoft Kinect). Even in those cases containing false positives, the duration of the algorithm error does not compromise the performance of a robust detection. Redundancy of sensors allowed us to track users also in critical position (e.g. two users aligned one behind the other in front of one camera).

A comparison with the literature shows that in [35] a similar multi-Kinect approach is developed. In this work, a point-cloud based approach in user detection is carried out, using point-cloud centroids to find correspondences between couple of users. With respect to that, approach in searching for users correspondence based on distances between RFs results in a lower threshold (0.4 m versus 0.6 m) and a reasonably higher accuracy for a couple of Kinects calibrated with comparable residuals (over 30 mm). The main limit in our approach is related to the user's envelope, which is based on spheres and cylinders constructed along the user's skeleton, whereas the work there presented uses point-clouds also for computing envelopes (leading to a user-dedicated envelope instead of a general one).

Latencies of `STOP` command (Figure 4.5 in Chapter 4) after the detection of a possible collision (i.e. the robot hitting the user's envelope) showed the ability to assert the stop of robot motion whithin a delay smaller than 2 ms. This leads to a sub-millimetric TCP displacement before the beginning of the braking procedure for every considered velocity.

Regarding the behavioural control of the robot, results showed that it is possible to robustly change the velocity of the robot during its movement (Figure 4.6 in Chapter 4) without noticeable TCP oscillations. We implemented a simple

control that slows or stops the robot movement when an emergency situation is detected. After the `STOP` assertion, the movement is re-started only after the acknowledgment of the emergency by a user. The braking performance showed a over-millimetric displacement (in terms of RMSE, Figure 4.8 in Chapter 4) when moving at velocities higher than 1.0 cm/s even if the commanded position sent to the robot shows a null displacement (Figure 4.7 in Chapter 4). A comparison with the literature shows that in [40] an analysis of movement accuracy of the same robot is carried out. Results showed a comparable accuracy in position when moving at the same velocities. In particular, the same dependance on the velocity of movement is enhanced. For our safety purpose, the choice of a slow velocity (i.e. 1.0 cm/s) when moving near to the entry target and a faster velocity (e.g. 7.5 cm/s) elsewhere can ensure sub-millimetric displacement in reaction to a `STOP` event inside a critical area, while maintaining a reasonable approaching velocity for almost the whole procedure.

The architecture here presented resulted to be able to supervise a semi-autonomous robot during a target approaching procedure (e.g. the preliminary phase of a SEEG), ensuring robust intraoperative tracking and surgical staff detection and tracking, as well as a reaction to particular emergency situations that may arise. Latencies of reaction at the stated velocities of task execution leads to sub-millimetric displacement (or untracked movement) of the robot TCP, which are suitable for neurosurgical applications [42].

Regarding the human detection algorithm, future works may be related to the development of a detection that handles a user-defined envelope instead of a generally defined ones. OpenNI algorithm `Xn::getUsersPixels`, that should return pixels belonging to a user identified by a set of RF, will be investigated. Moreover, since the RF-based approach for searching correspondences showed a better performance with respect to the point-cloud-only approach, a fusion may be implemented in which correspondences are computed over users RF and results

are merged using point clouds. Since point-cloud based algorithms are computationally heavy, GPGPU programming may be suitable for this purpose.

Regarding the robot behavioural control, future works may involve the development of a more complex behaviour, for example including an *hands-on* cooperation when a user is detected to be in contact with the robot and the internal torque sensors register a particular force. Moreover, the robot can be sent directly near the hand of the surgeon in order to facilitate the *hands-on* procedure. Due to the criticity of these procedures, a dedicated Workflow Descriptor should be provided in order to check if the requested transition is allowed at the current step.

Bibliography

[1] P. Gomes. "Surgical robotics: Reviewing the past, analysing the present, imagining the future". In: *Robotics and Computer-Integrated Manufacturing* 27.2 (2011), pp. 261–266.

[2] T. A. Mattei et al. "Current state-of-the-art and future perspectives of robotic technology in neurosurgery". English. In: *Neurosurgical Review* 37.3 (2014), pp. 357–366. ISSN: 0344-5607. DOI: 10.1007/s10143-014-0540-z. URL: http://dx.doi.org/10.1007/s10143-014-0540-z.

[3] J. Cobb. "Hands-on Robotic Unicompartmental Knee Replacement". In: *Navigation and MIS in Orthopedic Surgery*. Ed. by J. B. Stiehl et al. Springer Berlin Heidelberg, 2007, pp. 284–296. ISBN: 978-3-540-36690-4. DOI: 10.1007/978-3-540-36691-1_37. URL: http://dx.doi.org/10.1007/978-3-540-36691-1_37.

[4] GP Moustris et al. "Evolution of autonomous and semi-autonomous robotic surgical systems: a review of the literature". In: *The International Journal of Medical Robotics and Computer Assisted Surgery* 7.4 (2011), pp. 375–392.

[5] E.A. Sisbot et al. "Implementing a Human-Aware Robot System". In: *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*. 2006, pp. 727–732. DOI: 10.1109/ROMAN.2006.314487.

[6] G. Soros et al. "A cognitive robot supervision system". In: *Applied Machine Intelligence and Informatics, 2009. SAMI 2009. 7th International Symposium on*. 2009, pp. 51–55. DOI: 10.1109/SAMI.2009.4956637.

[7] R.H. Taylor and D. Stoianovici. "Medical robotics in computer-integrated surgery". In: *Robotics and Automation, IEEE Transactions on* 19.5 (2003), pp. 765–781. ISSN: 1042-296X. DOI: 10.1109/TRA.2003.817058.

[8] H. Monnich, H. Worn, and D. Stein. "OP sense—A robotic research platform for telemanipulated and automatic computer assisted surgery". In: *Advanced Motion Control (AMC), 2012 12th IEEE International Workshop on*. IEEE. 2012, pp. 1–6.

[9] M. D. Comparetti et al. “Event-based device-behavior switching in surgical human-robot interaction”. In: *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA, Hong Kong)*. Accepted. 2014.

[10] C. Y. Baldwin and K. B. Clark. *Design Rules, vol.1: The Power of Modularity*. 1st ed. Massachusetts Institutes of Technology, 2000.

[11] P. Clements and F. Bachmann. *Documenting Software Architectures: Views and Beyond*. 2nd ed. Addison-Wesley Professional, 2010.

[12] T. D. Miller and P. Elgar. “Defining Modules, Modularity and Modularization”. In: *Proceedings of 13th IPS Research Seminar*. Ed. by Aalborg University. 1998.

[13] C. Zielinski. “Formal approach to the design of robot programming frameworks: the behavioural control case”. In: *Bulletin of the Polish Academy of Sciences* 53 (2005), pp. 57–67.

[14] A. Bonarini et al. “R2P: An open source hardware and software modular approach to robot prototyping”. In: *Robotics and Autonomous Systems* (2013, in press).

[15] G. S. Guthart and J. K. Salisbury. “The Intuitive Telesurgery System: Overview and Application”. In: *Proceedings of 2000 IEEE, International Conference on Robotics and Automation*. Ed. by IEEE. 2000.

[16] J. Dumpert et al. “Semi-autonomous Surgical Task Using a Miniature In vivo Surgical Robot”. In: *31st Annual International Conference of the IEEE EMBS* (2009).

[17] M. Loetzsch, M. Risler, and M. Jungel. “XABSL - A Pragmatic Approach to Behaviour Engineering”. In: *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006).

[18] A. Kent and J. Williams. *Encyclopedia of Computer Science and Technology, Vol. 25*. 1st ed. Marcel Dekker, Inc., 1995.

[19] *UML state machine*. URL: `http://en.wikipedia.org/wiki/UML_state_machine` (visited on 06/23/2014).

[20] J. Burgner et al. “Ex-vivo accuracy evaluation for robot assisted laser bone ablation”. In: *Int. J. of Medical Robotics and Computer Assisted Surgery* 6 (2010), pp. 489–500.

[21] A. Agovic et al. “Haptic Interface Design Considerations for Scrub Nurse Robots in Microsurgery”. In: *Proceedings of the 18th Mediterranean Conference on Control and Automation* (2010).

[22] R. Bischoff et al. “The KUKA-DLR Lightweight Robot arm - a new reference platform for robotics research and manufacturing”. In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. 2010, pp. 1–8.

[23] KUKA Roboter GmbH. *KUKA System Software. Reference Manual.* 2010.

[24] Norther Digital Inc. *Optotrak Certus User Guide (s-Type)*. 2006.

[25] Norther Digital Inc. *Polaris Vicra User Guide.* 2005.

[26] Y. Arieli et al. *Depth mapping using projected patterns.* US Patent 8,150,142. 2012.

[27] K. Khoshelham and S. O. Elberink. “Accuracy and resolution of kinect depth data for indoor mapping applications”. In: *Sensors* 12.2 (2012), pp. 1437–1454.

[28] M. Quigley et al. “ROS: an open-source Robot Operating System”. In: *Proc. Open-Source Software Workshop Int. Conf. Robotics and Automation.* 2009.

[29] P. Soetens. *The Orocos Component Builder’s Manual, v.2.6.0.* Orocos Real-Time Toolkit, 2012. URL: `http://www.orocos.org/stable/documentation/rtt/v2.x/doc-xml/orocos-components-manual.html`.

[30] P. Soetens. *The Deployment Component, v.2.6.0.* Orocos Real-Time Toolkit, 2012. URL: `http://www.orocos.org/stable/documentation/ocl/v2.x/doc-xml/orocos-deployment.html`.

[31] M. Klotzbücher and H. Bruyninckx. “Coordinating robotic tasks and systems with rFSM Statecharts”. In: *JOSER: Journal of Software Engineering for Robotics* 3.1 (2012), pp. 28–56.

[32] D. Harel and A. Naamad. “The STATEMATE semantics of statecharts”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 5.4 (1996), pp. 293–333.

[33] M. Klotzbücher, P. Soetens, and H. Bruyninckx. “Orocos rtt-lua: an execution environment for building real-time robotic domain specific languages”. In: *International Workshop on Dynamic languages for RObotic and Sensors.* 2010, p. 284289.

[34] E.J. Henley and H. Kumamoto. *Reliability engineering and risk assessment.* Prentice-Hall, 1981. ISBN: 9780137722518. URL: `http://books.google.it/books?id=tvNTAAAAMAAJ`.

[35] T. Beyl et al. “Multi kinect people detection for intuitive and safe human robot cooperation in the operating room”. In: *Advanced Robotics (ICAR), 2013 16th International Conference on.* 2013, pp. 1–6. DOI: `10.1109/ICAR.2013.6766594`.

[36] R. Perrone et al. “Ontology-based modular architecture for surgical autonomous robots”. In: *6th Hamlyn Symposium*. 2014.

[37] E. De Momi et al. “EuRoSurge Workflow: From ontology to surgical task execution”. In: *Proceedings of the 2013 CRAS Workshop* (2013).

[38] R. Perrone et al. “From ontological knowledge to surgical task execution in the EuRoSurge experiment”. In: *Proceedings of the 2013 IROS Conference* (2013).

[39] M. Klotzbücher, G. Biggs, and H. Bruyninckx. “Pure Coordination using the Coordinator–Configurator Pattern”. In: *arXiv preprint arXiv:1303.0066* (2013).

[40] M. D. Comparetti. “High level control of robot behavior in neurosurgery”. PhD thesis. Politecnico di Milano, 2014.

[41] G. Schreiber, A. Stemmer, and R. Bischoff. “The fast research interface for the kuka lightweight robot”. In: *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010)*. 2010.

[42] R. R. Shamir et al. “Localization and registration accuracy in image guided neurosurgery: a clinical study”. In: *International journal of computer assisted radiology and surgery* 4.1 (2009), pp. 45–52.

[43] M. D. Comparetti et al. “Accurate multi-robot targeting for keyhole neurosurgery based on external sensor monitoring”. In: *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine* 226.5 (2012), pp. 347–359.

Written in LaTeX.

©Federico Nessi, *2014*.