UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Anastasia Bolotnikova

Articulated Object Tracking from Visual Sensory Data for Robotic Manipulation

Master's thesis (30 ECTP)

Supervisors: Assoc. Prof. Gholamreza Anbarjafari
Prof. Abderrahmane Kheddar

Tartu 2017

Visuaalsel informatsioonil põhinev roboti juhtimine liigestatud objekti manipuleerimisel

Lühikokkuvõte:

Roboti juhtimine liigestatud objekti manipuleerimisel vajab robustset ja täpset objekti oleku hindamist. Oleku hindamise tulemust kasutatakse tagasisidena vastavate roboti liigutuste arvutamisel soovitud manipulatsiooni tulemuse saavutamiseks. Selles töös uuritakse robootilise manipuleerimise visuaalse tagasiside teostamist. Tehisnägemisele põhinevat servode liigutamist juhitakse ruutplaneerimise raamistikus võimaldamaks humanoidsel robotil läbi viia objekti manipulatsiooni. Esitletakse tehisnägemisel põhinevat liigestatud objekti oleku hindamise meetodit. Me näitame väljapakutud meetodi efektiivsust mitmel erinevatel eksperimendil HRP-4 humanoidse robotiga. Teeme ettepaneku ühendada masinõppe ja serva tuvastamise tehnikad liigestatud objekti manipuleerimise markeerimata visuaalse tagasiside teostamiseks reaalajas.

Võtmesõnad:

Liigestatud objekti jälgimine, Tehisnägemine, Ruutplaneerimine, Robootiline manipuleerimine, Visuaalsel informatsioonil põhinev liigutuste kontrol, Masinõpe

Articulated Object Tracking from Visual Sensory Data for Robotic Manipulation

Abstract:

In order for a robot to manipulate an articulated object, it needs to know its state (i.e. its pose); that is to say: where and in which configuration it is. The result of the object's state estimation is to be provided as a feedback to the control to compute appropriate robot motion and achieve the desired manipulation outcome. This is the main topic of this thesis, where articulated object state estimation is solved using visual feedback. Vision based servoing is implemented in a Quadratic Programming task space control framework to enable humanoid robot to perform articulated objects manipulation. We thoroughly developed our methodology for vision based articulated object state estimation on these bases. We demonstrate its efficiency by assessing it on several real experiments involving the HRP-4 humanoid robot. We also propose to combine machine learning and edge extraction techniques to achieve markerless, real-time and robust visual feedback for articulated object manipulation.

Keywords:

Articulated Motion Tracking, Computer Vision, Quadratic Programming, Robotic Manipulation, Vision Based Motion Control, Machine Learning

Acknowledgements

I would first like to express my gratitude to my thesis supervisors Assoc. Prof. Gholamreza Anbarjafari at University of Taru and Prof. Abderrahmane Kheddar at CNRS-LIRMM for helping me to navigate during the process of development of this thesis.

A very special gratitude goes out to my labmates Iiris Lüsi, Antonio Paolillo, Kévin Chappellet for a great help and many late nights staying with me in the lab. Thanks to all the members of research groups, where I was privileged to work, namely, the Intelligent Computer Vision research group at University of Tartu, Estonia and the Interactive Digital Humans research group at CNRS-LIRMM, Montpellier, France.

I would also like to thank Skype and Study IT in Estonia programme and IT Akadeemia Speciality Scholarship for support of my master studies. Also my gratitude goes out to the Erasmus+ program, which gave me an amazing opportunity to travel and conduct research for this thesis in LIRMM.

Last, but not least I want to thank professors and staff of the Institute of Computer science at University of Tartu for organization and execution of the master program, which I was fortunate to participate in.

Contents

Introduction

Modern robotic technology is no longer confined to the field of automation and structured environments. Indeed, robotics have spread into various services. Bill Gates forecast a robot in every home and a revolution that shares various commonalities to that of computers years ago. By being applied to various services, modern robotics has slowly and surely changed to be human-centric and is to be considered as part of the information and communication technologies, which transform the numerical data into actions in the real world. Now robots share the same space as humans and shall be considered as work partners (cobots), home companions (domotics), etc.

To be fully operational, robots should be able to use the same infrastructures as humans (anthropomorphic design), interact with humans (cognition) and manipulate the same tools and implements, e.g. house implements, as humans (dexterity). Contrary to humans, however, robots process strictly numerical data, and instructions such as "open a drawer" or "open a door" should be programmed for the robot to translate into planning and control that only process data of the sort "open the drawer that is at $\boldsymbol{X}$ position and $\boldsymbol{Y}$ orientation, by $\boldsymbol{Z}$ centimetres" or "open the door by $\boldsymbol{\pi/4}$ radians". Whereas the robot controls its motions using transducers (i.e. electronic sensors) such as joint optical encoders, force sensors etc., such equivalent sensors are not available on the house implements and any daily articulated objects that we manipulate in general. It is important to note that, from a robotic perspective, both daily objects and humans can be modelled as articulated structures. Thus, in order for the robot to interact with a human or help a frail person by assisting her/his motion, it has to know the exact location and the posture of the person so that it plans the location of touch and the manner of contact and applies force accordingly. Obviously humans also cannot be embedded by invasive sensors only to allow the robot to understand their state and their intentions by inferring their posture or more generally their motion. This brings us to the conclusion that other information acquisition sources, such as, for example, video cameras, must be used to understand the state of manipulated objects.

Object manipulation is a very wide and important area of study in robotics. For a given application, set of tasks, and performances (e.g. in terms of execution time and reliability) the robotic platforms that are designed to physically interact

with surrounding environment need to possess good perceptual and reasoning capabilities to manipulate the objects skilfully. In modern manufacturing and in almost any other service robotic applications, computer vision is one of the most important artificial perception cues. Indeed, basic perceptual skills for robotic manipulation tasks are being studied by many branches of computer vision (CV), such as object segmentation, detection, recognition, pose estimation and visual tracking. In this thesis we use vision to allow the robot to extract the information required for motion planning and control about the objects it has to manipulate. As a use case, we regard daily object manipulation (e.g. drawer opening), and do not consider physical human-robot interaction at the moment. However, we keep in mind that continuation of this work should be potentially extendible to the case of more complex articulated structures, such as humans. We also assume that we know what objects we will ask the robot to manipulate or operate and focus our research on poly-articulated objects.

Vision is also used in robotics to control the motion. Indeed, Visual Servoing (VS) consists in incorporating the information extracted from the visual data into the planning and the control law by which robot motion is generated. As an academic example of such a control law, we can refer to the block diagram in Figure 1. Here the task of manipulating one degree-of-freedom (DoF) articulated object (e.g. door, drawer) is considered. Of course, our work is not limited to this restriction, which is chosen only to convey the ideas. For a robot, the manipulation task is defined in terms of the desired configuration of an object, $\boldsymbol{\theta_d}$, at the end of manipulation. For example, "open drawer by **20**cm" would be formulated in the control as $\boldsymbol{\theta_d} = \mathbf{20}$cm. The completion of the task is regulated by the value of the error $\boldsymbol{\epsilon}$ between the desired configuration, $\boldsymbol{\theta_d}$, and the actual current configuration of the object, $\boldsymbol{\theta_m}$, that we cannot access directly. According to the control law illustrated in Figure 1, the current state value measure $\hat{\boldsymbol{\theta}}_{\boldsymbol{m}}$, obtained from the perception measurement block **M** (ideally $\hat{\boldsymbol{\theta}}_{\boldsymbol{m}} = \boldsymbol{\theta_m}$), is compared to $\boldsymbol{\theta_d}$. The error value $\boldsymbol{\epsilon}$ is computed and processed by the controller block **C** to generate appropriate input signals $\boldsymbol{u}$ for the motion update of the robotic system **S**, so that the value of $\boldsymbol{\epsilon}$ is driven to zero as a result of the closed-loop control.

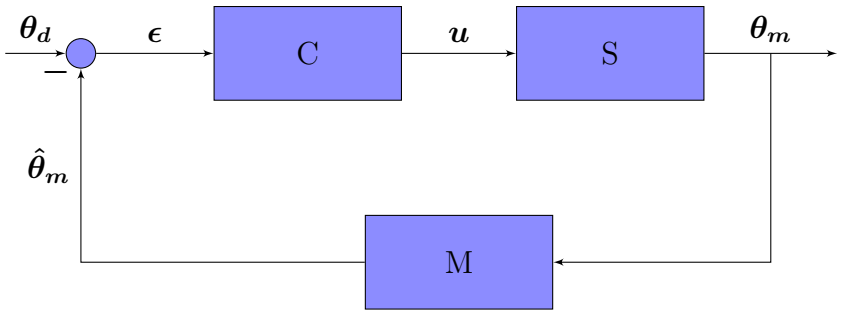


Figure 1: Closed-loop control law or feedback controller.

The work in this thesis is mainly concerned with the block **M** of Figure 1 diagram, which is responsible for estimating the value of $\boldsymbol{\theta_m}$. We show that if we consider articulated objects to be modelled as other "robots" with passive joints (i.e. the objects and their parts have no actuators to change their configuration and also no encoders to measure the latter changes should they occur from an external force/action), we can estimate $\boldsymbol{\theta_m}$ from the robot embedded vision and integrate the results in a multi-robot task-space based controller formulated as Quadric Programming (QP), which controls altogether the robot and the articulated object that is to be manipulated or operated.

Let us now highlight all the previous concepts through a concrete example. The robotic platform used for the object manipulation in this work is the HRP-4 humanoid robot illustrated in Figure 2, together with one of the tasks that is considered in our work: opening the printer's drawer.

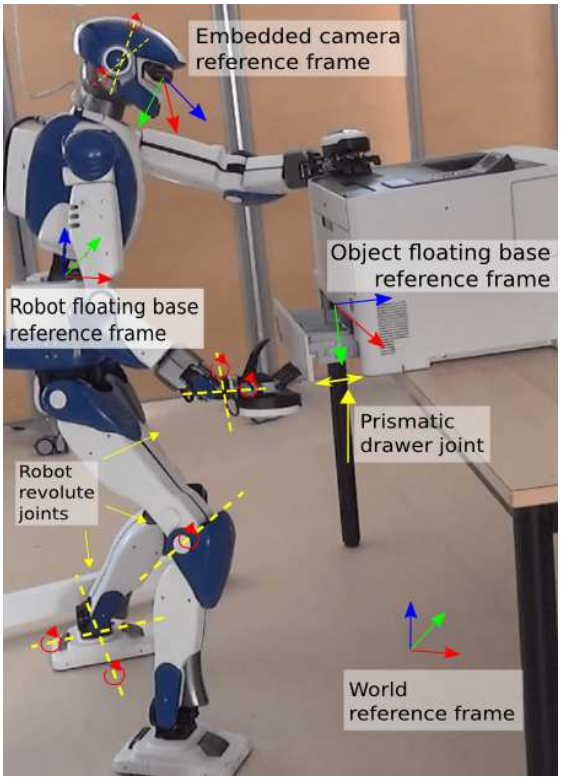


Figure 2: The HRP-4 humanoid robot manipulating an articulated object (the printer). Illustration of floating base frames and some joints.

In the example of Figure 2, the HRP-4 motion is driven by a task-space QP controller, in which the goals to be achieved are written as errors in the task-space. The QP controller computes desired joint accelerations, desired contact forces, and eventually desired torques for the robot to drive the tasks to their desired state – or equivalently, to drive the task errors to zero. Since the robot interacts with other objects that are articulated in the general case, they are

seen as additional "robots", whose models are incorporated and used by the QP together with that of the robot (HRP-4). For instance, the printer is modelled as a "robot" having a floating base with position and orientation degrees of freedom in 3D space (indicated as object floating base frame in Figure 2) and a prismatic joint (that of the drawer). The configuration of the robot HRP-4 means where it is (position and orientation w.r.t world frame), and the joints articulations say what is its actual posture. The configuration of the "robot" printer means where is it situated w.r.t robot's camera frame or the world frame; its joint configuration in this case means how much the drawer is opened. However, contrary to the robotic systems that are embedded with encoders or other sensors that measure directly the value of their current configuration (position, orientation and joints), the printer does not have any encoders, and the opening value of the drawer (which acceleration is a decision variable of the QP), as well as its position and orientation, is to be determined from the robot embedded sensors: that is, the HRP-4 camera, and eventually the HRP-4 encoders once contact constraint between the drawer and the robot's gripper is established.

The novelty of our work is to design and implement a framework for estimating the configuration of the poly-articulated objects from vision, when manipulated by the actuated robots. This then allows us to effectively close the control loop in order for the robot to achieve the desired goals with the articulated object of interest. Our main contributions include the following aspects:

- A study of features and visual sensory data types for the articulated object tracking and pose estimation;
- Comprehension and implementation of the vision based motion control in the multi-robot Quadratic Programming controller framework;
- Incorporation of the knowledge about the articulated structure of the object into the tracking framework;
- Articulated object state estimation from points and edge image features;
- Evaluation of the proposed system in challenging scenarios, such as drawer opening and pulling circuit breakers of a mock-up provided by the Airbus.

We structured the thesis as follows: Chapter 1 gives an overview of the research field in articulated object tracking. In the conclusion of Chapter 1, the main challenges of the articulation tracking for robotic manipulation and drawbacks of the existing methods are outlined. Chapter 2 presents the methods used for vision based motion control in the QP controller framework. Chapter 3 describes the details of the proposed articulated object configuration estimation, which takes advantage of visual servoing and image processing principles. In Chapter 4 the experimental results are presented.

Chapter 1

Object tracking: a brief overview

1.1 State of the Art: overview and evaluation of existing methods

It is important to first screen existing work and research efforts in object tracking should they be adaptable to solve our problem of providing visual feedback on the state of an object for use in real-time control of robotic manipulators in real-life experimental settings.

This chapter reviews the existing methods that we identified as potentially useful for our needs in tracking passive articulated objects (i.e. continuous estimation of their pose in 3D and eventually their joints values) to close the loop in the multi-robot QP controller. We selected these approached on the basis of (i) real-time performance, and (ii) potential use for the control of robotic manipulation. We sort the reviewed approaches into 3 main categories:

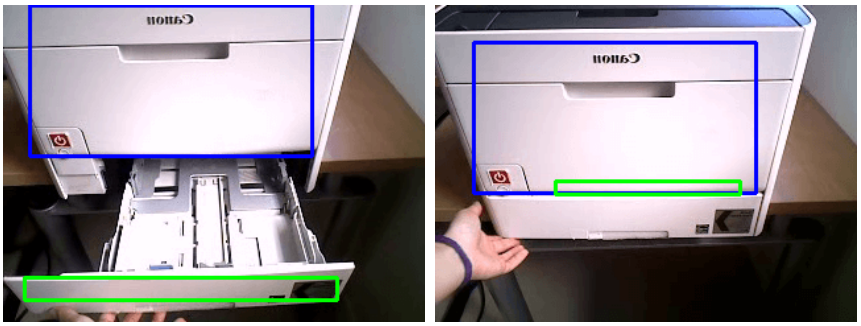
- Machine Learning based tracking (Section 1.2),
- Stochastic and gradient-free tracking (Section 1.3),
- Control and gradient based tracking (Section 1.4);

Note that in this section we bring a brief and general overview of the methods, concentrating more on the computer vision part of the methodologies for visual tracking and its advantages and drawbacks for the use in the robotic manipulation control context. The conclusions, made after the *state-of-the-art* overview, are explained in detail in Section 1.5 of this chapter. We review the articulation tracking background, specifically concentrating on joint value reconstruction later in this work in Section 3.3 of Chapter 3.

1.2 Machine Learning based tracking

A popular and efficient strategy for object tracking is to use the advances in machine learning (ML) algorithms to quickly detect and track specific patterns of the object in the video stream [1][2][3]. The patterns of the object parts can be learned beforehand, and the trained models can be used afterwards in the detection and tracking processes. However, in the case where an object does not have any specific pattern (e.g. the plain white face of a printer), the use of ML in computer vision becomes more challenging.

The main advantage of ML based approaches is real-time performance. The beginning of the real-time ML based object detection era can be associated with the *Viola-Jones framework* proposed in 2001 by Paul Viola and Michael Jones [2]. They proposed to learn the Haar-feature cascade of weighted simple features to learn complex patterns from the data with the AbaBoost algorithm. A significant increase in the speed of both training and detection was achieved due to the utilisation of integral image in the proposed framework [4]. In order to evaluate the potential of using the trained model for the articulated object part tracking, we trained the Haar cascade with the AdaBoost method on some sample data of the printer base and the front face of the printer drawer. To train the Haar cascade, frames from one of the sample videos were manually labelled, specifying the location of the upper static part of the printer front face and the lower front face of the articulated drawer. Figure 1.1 shows various frames, where the parts of the printer were automatically detected using the trained model of the Viola-Jones Haar cascade. The training process took approximately 7-11 hours on a standard CPU.



(a) Parts of the printer correctly detected

(b) Incorrect detection of the drawer location

Figure 1.1: Identifying the location of different articulated parts of the tracked object with trained Viola-Jones model.

To sum-up the detection performance of the trained Haar cascade, the following can be stated: successful detection of the upper static part of the front face was more stable compared to the detection of the drawer front face. The reason for

such a result could be the fact that the upper part of the printer contains more specific patterns (e.g. the on/off button, the printer logo), which are easier for the cascade of Haar features to learn. The overall performance, however, was not sufficient for a robust and smooth tracking of the articulated object parts. On top of that, the bounding-box type detection is not suitable for tracking in robotic manipulation, due to the fact that such detection does not provide sufficient geometric information about the detected object (e.g. orientation and exact pose are not known with sufficient accuracy in order to have stable closed-loop control). However, such a method of fast bounding-box type detection is quite useful for reducing the size of the entire input to the small sub-window containing the object of interest.

Much more sophisticated ML methods for computer vision were developed in recent years. The histogram of gradient descent (HOG) based bounding box type detection of objects greatly outperforms Haar feature cascade based approach[5]. We take advantage of this method to perform coarse object detection in our markerless tracking approach presented later in Section 3.7. The convolutional neural networks have gained great popularity due to much higher robustness to changes in pattern appearance, however, slightly less robust, but more simple methods, such as the HOG, proved to require much less time to train the model and perform detection.

1.3 Stochastic and gradient-free approaches

1.3.1 Blocks World Robotic Vision Toolbox

Scale-invariant feature transform (SIFT) [6] and a Particle Filtering [7] based approach for 6D pose tracking called BLORT has been proposed in 2010 by T. Morwald *et al.* [8]. The method has been implemented in the ROS framework[9] and made publicly available[1]. However, the BLORT module has not been maintained lately and is not supported in the new versions of ROS.

Prior to tracking, the BLORT method requires a user to map the textured surface of the object to the object model to extract SIFT features, which are used later in the matching process for the pose estimation. Offline mapping is performed by aligning the object with the projected object model and saving aligned sample frames.

It is important to note that SIFT features are expensive to compute and, depending on the number of features per frame, they may not provide real-time performance. Alternative feature descriptors have been proposed over the years, such as Speeded Up Robust Features (SURF) [10] or Oriented FAST and Rotated

[1] `https://github.com/pal-robotics/perception_blort`

BRIEF (ORB) [11]. The SURF and ORB feature descriptors are faster to compute, but at the same time, they are less robust to scale and orientation variations w.r.t. the SIFT. Moreover, none of the mentioned feature descriptors is applicable to the case of non-textured objects.

Another software, similar to BLORT, has been proposed by Karl Pauwels and Danica Kragic in 2015 [12]. This is an example of more recent work; the codes of the software are publicly available[2].

This software offers better scalability in terms of number of tracked objects thanks to the high degree of parallelism in the code implementation, which also allows us to efficiently fuse color and depth features for pose estimation and tracking. However, yet again, only textured objects can be tracked robustly due to the fact that SIFT features are used for object detection.

1.3.2 Depth Based Tracking via Particle Swarm Optimization

Depth information based object tracking approaches have been extensively studied. A considerably good performance has been achieved in the line of works [13][14][15][16]. Both rigid and articulated object tracking have been developed, as well as tracking of the interaction between various types of objects. Objects are modelled as rigid or articulated structures, which consist of some geometric primitives of a known size and shape. In these works the base for the object configuration (i.e. position, orientation and joint values) reconstruction is done using the Particle Swarm Optimisation (PSO), which is an evolutionary algorithm (EA) that does not impose the restriction of differentiability to the problem definition of configuration estimation. Those methods have shown to be highly robust and capable of automatic initialization and reinitialization in many various scenarios including manipulation of virtual objects [17] or force sensing from vision [18][19]. However, in the case when optimisation is done in a gradient-free fashion, it is impossible to prove or guarantee correct tracker error convergence. Thus, a gradient based approach would be the more safe and reasonable choice for the control of real object manipulation.

1.3.3 Robust edge and keypoint based tracking

The MBT framework, described previously in Section 1.4.1, can in practice lose the tracking relatively easily due to the fact that in the tracking loop it only considers a single hypothesis of the object pose. This leads to a number of issues. First of all, frame-to-frame pose estimations can be somewhat inconsistent and not always smooth. Secondly, if the number of features for the current hypothesis

[2] `https://github.com/karlpauwels/simtrack`

is insufficient to estimate the pose, the tracking is terminated and cannot be recovered.

To address those issues, a more robust and thus practically more useful edge and keypoint based object tracking framework was proposed by C. Choi *et al.* [20] in 2012. In [20], multiple hypotheses are considered for tracking a single object using a Particle Filter based architecture. The weighed sum of all the hypotheses (or particles) is then reported to be the true estimated pose of the object. In such a framework, the complete failure of one or even several hypotheses would not affect substantially the final pose estimation result, as long as there is a good amount of good pose hypotheses.

In [20], they also implemented automatic keypoint based initialization and reinitialization methods for the start of the tracking procedure, which is yet another advantage of the framework, which makes it practically more useful.

1.3.4 3D object tracking: an edge-based approach

In the same year of 2012, C. Choi *et al.* proposed an extension of their previous work for the case of non-textured objects [21]. In this work, the same principles of Particle Filtering with annealing on the SE**(3)** group are used to achieve robustness. Only edge features are used to compute the measurement likelihood and an efficient Fast Directional Chamfer Matching [22] is used for initialization with 49 edge templates obtained from the predefined polygonal mesh models of the tracked object. The sample code for this method is publicly available[3].

The two works of C. Choi *et al.*, described in this chapter, outline the robustness that can be achieved by using Particle Filtering. However, it is needless to say that considering multiple hypotheses (e.g. hundreds) instead of just one implies much higher computational complexity of the method. Nonetheless, it can also be noted that all the particles can be evaluated in parallel; thus computations in such an architecture can be accelerated by using the general-purpose graphics processing units (GPGPU) [23][24][25][26].

1.4 Control and gradient based approaches

1.4.1 Real-time markerless Model Based Tracking

A more suitable framework for robotic manipulation, which enables precise pose estimation from visual data, has been proposed in 2006 by A.I Comport *et al.* [27].

Later this framework has been implemented in the ViSP library as the MBT

[3] `https://github.com/CognitiveRobotics/object_tracking_2D`

(Model Based Tracking) module and made publicly available[4]. The MBT module of the ViSP library allows tracking rigid objects either with or without the texture on the surface. The tracking can be performed in three different modes:

- **Edge based tracking**, for objects without texture;
- **KLT keypoint tracking**, for textured objects when edges are not easily detectable;
- **Hybrid edge and keypoint based tracking**, which is the most robust of all three modes, if both texture and edge information are well acquired.

To perform rigid motion tracking, the CAD model of the object which is being tracked needs to be known and provided. The CAD model is then projected onto the image and determines the location of the contours which correspond to the edges of the object. Those contours are then tracked iteratively using the Moving Edge algorithm [28]. As the new position of the edge point and/or keypoint is only searched for in a limited neighbourhood of the previous location, there is a limitation on the maximum amount of inter-frame movement of the object. There is a possibility of increasing the size of the neighbourhood, but that, obviously, also means that more computations are to be performed.

In order to evaluate the performance of the MBT module on the sample printer data, two CAD models of the rigid front faces of the separate rigid parts of the printer have been defined. For the tracking to be more robust to noisy or missing information, additional (auxiliary) edges were also included in the model along with the edges that define the boundaries of the rigid parts. Figure 1.2 demonstrates the projection of the defined CAD models onto several frames from the sample video of the printer drawer opening/closing.

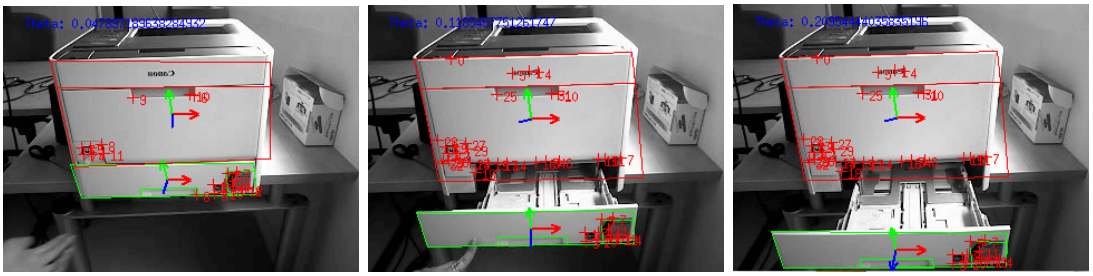


Figure 1.2: Separate tracking of different rigid parts of the articulated object with the MBT module while the object is being manipulated.

As can be seen from the sample frames from the video sequence, presented in Figure 1.2, tracking is much more accurate than in the case of Viola-Jones. The poses of the articulated parts are estimated with higher accuracy, and relative

[4] `https://visp.inria.fr/mbt/`

displacement can thus be computed to estimate the value of the drawer opening (i.e. the prismatic joint $\boldsymbol{\theta_m}$).

Originally, the MBT object tracking framework was proposed for the case of rigid body tracking. However, many articulated objects can be defined in terms of a physically constrained ensemble of rigid parts. Thus, the task of articulation tracking can be directly approached from the point-of-view of generalization and extension of rigid body tracking framework. This is discussed in more detail in Section 1.4.4.

1.4.2 Dense Articulated Real-time Tracking

A dense depth based approach for articulated object tracking has been proposed by T. Schmidt *et al.* [29]. This tracking framework is called DART. In this work GPGPU CUDA acceleration is used to provide real-time performance. The computational complexity, and thus the necessity to use GPU acceleration, in the proposed method is due to the decision to use dense depth data and not a sparse subset of the available data. The authors even made use of so called "negative" depth information, meaning that missing depth data was also incorporated into the energy function subject to minimization.

In the DART framework, signed distance functions (SDF) are generalized to the case of articulated objects to account for the physical constraints between object parts in the tracking process. The method requires a model of the tracked object to be defined in the XML format, clearly and precisely stating what is the geometry of each part, how parts are connected and positioned with respect to each other (as this is used in describing a robot). An initial guess also need to be provided with high accuracy in the DART framework to start the tracking process.

From the experiments conducted in this work with the publicly available code of the DART framework[5], only brief tracking of a simple slowly moving rigid object was achieved. When we tested the framework on a two-link articulated printer object model, we did not obtain good tracking performances. Only few seconds of correct tracking of the rigid object was achieved. Such results can be explained by insufficient accuracy in the initial guess manually provided on the first iteration of the tracking loop. Another possible obstacle for a better tracking could be a large amount of noise in the scene, which "confuses" the optimization procedure and leads to false pose estimation in a local minima of the objective function. Since the framework only considers a single hypothesis in the local optimizer, it is thus impossible to recover after the algorithm converges to a local minima or after an insufficiently accurate initial pose guess has been provided.

Nevertheless, effort toward using dense depth (hence featureless) tracking is still active area of research in the computer vision community, and we will consider

[5] `https://github.com/tschmidt23/dart`

investigating it more in future work.

1.4.3 Tracking of objects with identical appearance

The challenge of tracking multiple objects with identical and possibly non-textured appearance has been addressed in [30]. In this work a probabilistic approach is exploited in order to define the energy function that contains both visual sensory data and physical constraint terms in it. The energy function is then minimized via the Levenberg-Marquardt method [31].

The main idea of this method is to fuse the Signed Distance Functions (SDF) of all tracked objects into a single SDF. This tracking method can be extended to the case of multiple objects without a significant increase in the computations, due to the fact that optimization can still be performed once for a single function that describes the geometry of multiple tracked objects. The physical constraints are added to the energy function to ensure that the estimated poses of independently moving objects do not occupy the same space. However, the objects are assumed to be moving independently in the probabilistic model of the tracker, and no articulation related constraints are modelled in the framework. The implementation of this method can achieve over 80 FPS speed performance on a CUDA-capable GPU and has been made publicly available since 2014[6].

1.4.4 Kinematic Sets

Several frameworks and formalisms for articulated motion tracking have been proposed recently, e.g. [32][33][34]. The tracking of the articulated motion is typically formulated as a error minimization task for the error between the observations ($\boldsymbol{s^*}$) and the forward-projection of the object model ($\boldsymbol{s(x)}$), where $\boldsymbol{x}$ defines the configuration of an object which needs to be estimated (either the pose $\boldsymbol{r}$ of a rigid body or the minimal set of parameters $\boldsymbol{q}$ in the case of articulated motion):

$$\boldsymbol{\delta} = \sum(\boldsymbol{s(x)} - \boldsymbol{s*}) \tag{1.1}$$

In 2004 a method for poly-articulated object tracking was proposed in [34]. This method is an extension of the rigid body motion tracking method proposed in [27] and partially described previously in Subsection 1.4.1. Both [27] and [34] use the Virtual Visual Servoing [35] method to iteratively minimize the error defined in 1.1, by deriving a robust control law that drives the virtual system, defined by $\boldsymbol{x}$, in the direction where error converges to a locally lowest possible value. Similarly to how this is done in visual servoing, where the camera position is controlled and physically moved into the direction minimizing the error between where the object is in the frame, $\boldsymbol{(u, v)}$ coordinates in the image, and where is should be.

[6] `http://www.robots.ox.ac.uk/~victor/libisr/index.html`

As a result of the minimization procedure, the best estimate of $\boldsymbol{s(x)}$ is found and the corresponding configuration of the virtual system, $\boldsymbol{x}$, is said to be the current true state of the system.

The robust control law to estimate rigid body motion derived in [27] is:

$$\boldsymbol{v} = -\boldsymbol{\lambda}(\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}_{\boldsymbol{s}})^{+}\boldsymbol{D}(\boldsymbol{s}(\boldsymbol{r}) - \boldsymbol{s}^{*}) \tag{1.2}$$

where $\boldsymbol{v}$ is the velocity of the virtual camera; $\boldsymbol{r}$ is the position and orientation (pose) of the virtual camera; $\boldsymbol{D}$ is a weight matrix to reduce the influence of the outliers and; $\boldsymbol{L_s}$ is the interaction matrix between the visual feature velocities, $\dot{\boldsymbol{s}}$, and the camera velocity, $\boldsymbol{v}$. For more details on the control law derivation, the interested reader may refer to the original paper.

The control law for the articulated object tracking derived in [34] is:

$$\boldsymbol{gpv} = -\boldsymbol{\lambda}(\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}_{\boldsymbol{D}}\hat{\boldsymbol{A}})^{+}\boldsymbol{D}(\boldsymbol{s}(\boldsymbol{q}) - \boldsymbol{s}^{*}) \tag{1.3}$$

As can be noted, the difference between the equations 1.2 and 1.3 lies mainly in:

1. the definition of the system state ($\boldsymbol{r}$ is the camera pose in the case of a single rigid motion tracking; whereas $\boldsymbol{gpv}$ is is a general parameter vector of common and intersecting velocities in the case of the articulated object);
2. the articulation matrix $\boldsymbol{A}$. This matrix defines how $\boldsymbol{gpv}$ is related to the velocities of every rigid part in the articulated mechanical structure of the tracked object.

1.5 Main challenges

1.5.1 Real-time requirement

In order to use the object detection and tracking in control, we need to account for various challenges. The first challenge comes from the requirement of the control framework to perform all computations in **real-time** (i.e. within or less than the QP control-loop period to be used by the controller). To extract useful information for object detection, it requires processing a large amount of data (4 channels in case of RGB-D data $\times$ the resolution of the camera) possibly several times in one iteration at a frame rate, which in the case of a standard real-time system is at least 25 FPS. That leaves on average only 40ms to fully process the current frame.

The problem becomes even more sensitive to real-time, when the robotic context and its corresponding motion planing issues are considered, due to the fact that real-time motion planing, for example using the QP control framework, typically

runs at a 200 FPS rate. In this case, the vision frame should ideally be processed in the order of a millisecond (under 5ms). It is important to note that, when using a standard camera, an additional problem occurs because frame acquisition rate limits slow down the process of visual information processing. A typical camera can provide 30-60 FPS frame acquisition rate. That means that, even if image processing algorithm is finished in under 5ms, for the next several dozens of millisecond there is no new vision frame acquired by the camera for image processing to act on it.

The real-time requirement challenge can partially be solved by performing some computations in parallel with the help of modern GPGPUs, as a large part of the image processing tasks for object detection can be parallelized and are GPU friendly (e.g. processing various subparts of the image, evaluating several position hypotheses, etc.). However in practice still, not many systems or robotic platforms are equipped with such powerful processing units. The real-time requirement becomes even more crucial in the context of robotic physical interaction, due to the fact that all computations for the motion update are usually carried out on the same platform; thus a limited amount of processing power is available for the perception related computations.

1.5.2 Data inconsistency and occlusions

Visual sensory data can be very unstable and inconsistent in a video sequence. The RGB data that encodes the color information for the objects in the scene always varies significantly with slight changes in illumination of the scene or part of the scene (e.g. shadows, reflected light rays). The development of a very robust and completely illumination-invariant color or intensity based object detector is extremely difficult. In practice, some assumptions about lighting are made when using color information in the detection algorithm.

The depth information which can be acquired by the range sensors, such as the Kinect or Xtion, is less sensitive to the changes of light in the scene. However, another challenge arises, as the depth data can be highly inconsistent and noisy at the edges of the objects and outside the distance of use (too close or too far from the camera). Missing depth data can be to some degree recovered by the means of interpolation, and noise can be partially eliminated by outlier removal.

Inconsistency and noisiness of the visual sensory data requires the implementation of pre-processing operations and statistical outlier detection and elimination before low or high level visual features can be extracted from the frame. That increases the overall computation complexity of any detection algorithm, but gives the chance to increase robustness of the method, which makes it more useful and generic in practice.

The issue of partial or full occlusion of the tracked objects make already challenging

visual data even more complex to work with. In the case of full occlusions, the recovery from tracking failures and reinitialization of the tracker is required. More details on reinitialization and recovery are given later in Section 1.5.4.

In the case of partial occlusions of the tracked object, the detection algorithm is required to be flexible and robust in terms of adaptation to new conditions in data availability. Ideally the relative position between seen and unseen parts of the object needs to be exploited efficiently to handle partial occlusions well.

1.5.3 Pose estimation inconsistency

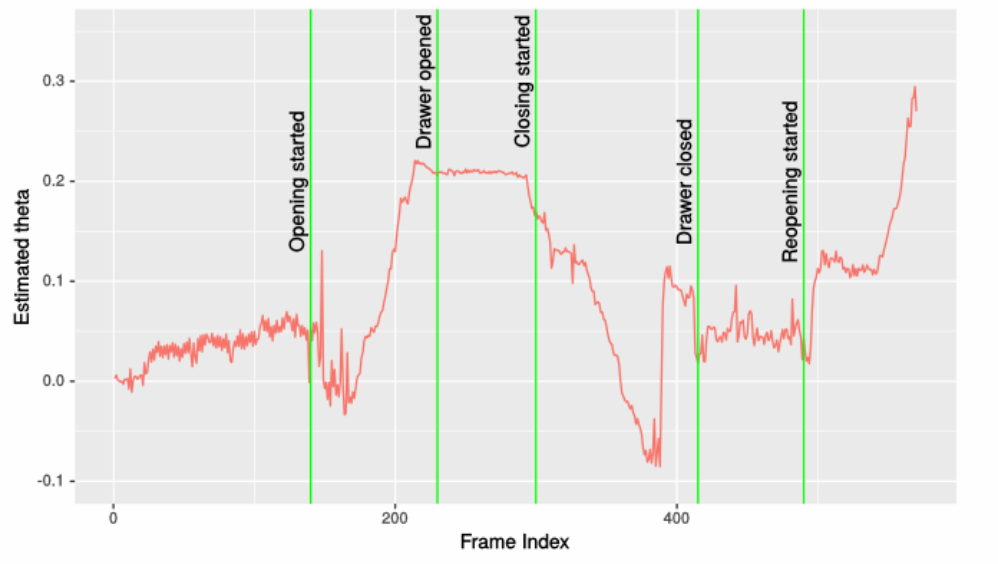
When estimating the current pose of the object from the data that can be noisy and inconsistent at times, as described in Section 1.5.2, the problem arises that the estimated poses for subsequent frames would also lead to abrupt results.

The problem of inconsistent and noisy frame-to-frame pose estimation can be particularly problematic in the context of a visual servoing feedback controller. Large inconsistencies in pose estimation, when fed directly into the motion control loop, will inevitably result in undesired edgy (or jerky), rough and inconsistent motions of the robot.

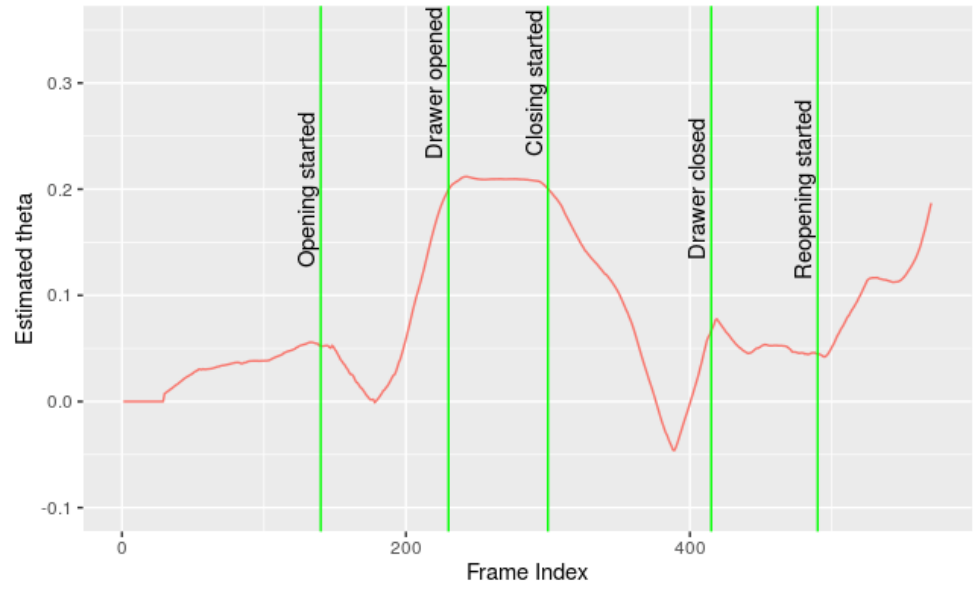
As an example of edgy pose estimation, we can look at the data collected from the experiments with a single hypothesis based MBT module described in Subsection 1.4.1. Figure 1.3a shows the values of $\boldsymbol{\theta_m}$ for the printer opening/closing manipulation experiment obtained by calculating relative displacement of the separate rigid parts tracked by the MBT module.

As can be seen from the plots, estimated values are rather noisy and inconsistent, as the actual process of manipulating the object was much smoother. The result of a mean filter applied to the $\boldsymbol{\theta_m}$ estimation, which would be practically more useful in the robotic manipulation context (but it introduces a phase, i.e. a delay), is shown in Figure 1.3b.

We can attempt to address this issue by filtering to the sequence of the estimated poses, for example by using a Particle Filter. Multiple randomly initialized hypotheses (or particles) of the current object pose estimate are considered, and the weighted average of those hypotheses is granted to be the actual pose of the object. This way the frame-to-frame pose estimation results become more consistent. The influence of such an approach is well illustrated in [20], described in Section 1.3.3. However, as we already mentioned, applying a smoothing filter to the data will induce a lag in the estimation and the data will always be retarded.



(a) Raw output of $\boldsymbol{\theta_m}$ estimation via the MBT module.



(b) Mean filter of window size 30 applied to the estimation of $\boldsymbol{\theta_m}$.

Figure 1.3: Examples of raw/noisy (a) and smoother (b) estimation of $\boldsymbol{\theta_m}$ parameter of the articulated object.

1.5.4 Initialization and recovery of the tracking

A basic scheme for pose estimation from 2D image features can be described by the block diagram shown in Figure 1.4.

This loop minimizes the error between measured and projected features to estimate the current pose of the object considering the previously known pose(s). The logic of the loop in Figure 1.4 is the following:

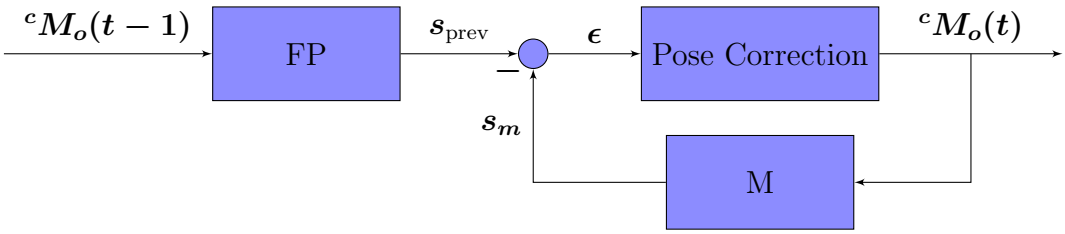


Figure 1.4: Pose estimation loop.

1. Get the previous pose estimation ${}^{c}\boldsymbol{M_o(t-1)}$;
2. Forward project (FP) the previous pose onto the new camera frame $\boldsymbol{f_t}$;
3. Search for most similar points $\boldsymbol{s_m}$ along the projected model contours $\boldsymbol{s}_{\text{prev}}$;
4. For each point found, determine the displacement, $\boldsymbol{\epsilon}$, from the corresponding point in the projected contour model;
5. Compute the new object pose to minimize $\boldsymbol{\epsilon}$;
6. Re-project the new pose to the same frame;
7. Go to 3 and repeat until $\boldsymbol{\epsilon}$ is nil.

Note that in the very first iteration of such an object tracking loop it is necessary to provide an *initial guess*, due to the fact that ${}^{c}\boldsymbol{M_o(t-1)}$ is simply non-existent when the algorithm just starts the tracking process (i.e. at $\boldsymbol{t=0}$). That implies yet another challenge for the object tracking framework. In some methods the initial guess has to be provided by the user, as it is done for MBT and DART described previously in this chapter. However, providing the initial pose manually is very tedious, not practical and also prone to errors. In practice of course, a given technology framework has to generate a "close to true" initial guess automatically with as little previous knowledge or supervision as possible. If the initial guess is not close to the true initial pose of the object, the tracking procedure will not be able to converge to the actual object state. That will cause a *tracking failure*, and the system would need to be able to *recover* from this situation and reinitialize the tracking.

Since the operations of initialization and recovery only have to be performed from time to time and not for each frame, it is possible to allow more computationally complex methods to be applied for addressing this challenge. In the process of initialization, the framework has to take raw images as an input and segment out from scratch where the object is, as well as predict the pose of this object. It is clear that in order to perform such (re)initialization, the framework needs to know at least "what is it looking for". Thus such information as a known *apriori* model of the object, specific patterns, etc. can be of good use for addressing the (re)initialization issue.

1.5.5 Handling of non-textured objects

When using the color intensity information, it is fairly simple to robustly track an object with a specific texture on its surface – or, what is a less desirable but still common practice, with distinctive markers. The distinctive texture can be efficiently and robustly matched using keypoints, as is done in [20], described previously in Subsection 1.3.3. In this case, precise initialization, recovery and robust tracking of the object become somewhat less challenging

A much more complex task is to track non-textured objects (or any object). In this case, only efficient exploitation of edge and depth information can allow us to achieve somewhat robust object tracking and precise (re)initialization.

1.6 Summary

To sum-up the evaluation of the existing approaches in the field of articulated object tracking, we bring out the *main criteria* that make the methodology practically useful in the context of robotic control. These are:

- The tracker must preform within the task-space control loop;
- Automated initialization of the tracking;
- Exploit possible *apriori* knowledge, but with as little supervision as possible, as for (re)initialization;
- Automate recovery from tracking failure and handle full or partial occlusions of the parts of tracked object (due to the robotic part overlap during reaching or manipulation);
- Use of joint constraint and limits to support articulation tracking (in addition, the kinematic of the robot in contact must be exploited);
- Exploit all available information, meaning that photometric and geometric information are properly fused and complement each other;
- Frame-to-frame pose predictions are smooth and do not cause jerky robot motions;
- Tracking of low- or non-textured objects.

By now, we have presented the overview in the field of computer vision based tracking of articulated objects. In the next chapter, we focus on the motion control part of the study conducted in this thesis.

Chapter 2

Vision based whole-body motion control in the Multi-robot QP framework

In the previous chapter, we looked into the computer vision field of research on articulated object tracking. This chapter is dedicated to the vision based motion control part (i.e. visual servoing) of the study conducted in this thesis.

2.1 Introduction

Before we decide on what and how visual information is extracted, processed and interpreted, first we look at how this information will be used in the low level motion control process. We need to define the methods to translate information extracted from the image into the update values for the robot joints. This in turn allows us to develop poly-articulated object configuration estimation using visual tracking techniques, which will be presented in the next Chapter 3.

In this chapter, the methods for vision-based motion control, or Visual Servoing (VS) [36][37], are described in the context of sample experiments using our Multi-Robot Quadratic Programming framework (QP for short) [38]. This functionality (i.e. describing robot goals or objectives as visual servoing tasks) was missing in the QP controller at the start of this thesis. Therefore, it was important to port it prior to conducting the experiments with the HRP-4 robot. First, we give explanations on the multi-objective whole-body humanoid motion control in the QP framework. Then, image-based and position-based visual servoing tasks are presented. Later in this chapter, several sample experiments with visual servoing in the QP are described in order to illustrate the vision based QP motion control in action. The full integration in a multi-modal force/vision control and the experimental results are described in Chapter 4.

2.2 Multi-objective task-space whole-body motion control via QP

Multi-objective, task-space control [39] formulated as QP appeared recently as a golden standard for the whole-body control of robots – hyper-redundant robots in general and humanoids in particular, see examples in [40][41][42][43][44][45][46][47]. In the QP control framework, tasks can be ordered in strict, weighted or hybrid priority. In a weighted (soft) priority scheme, which we are using, the tasks to be achieved at best are weighted according to their priority and summed up in the cost function of the QP. Those objectives, which need to be fulfilled strictly, are put in the constraint part of the QP. For instance, creating a contact is split into two parts: (i) a reaching contact task and (ii) a contact task. Reaching the contact is a task, which is defined as an error in the robot sensory space, that is put first in the cost function. As soon as the contact is established, this task is shifted into a QP set of constraints to maintain the contact. Tasks inclusions, removals and changes of priority are scheduled and managed by a finite state machine; see [47], which will be detailed with the integration and experiment in Chaper 4. To be self-contained, we recall here the main ingredients of the QP control framework, with the classical tasks and the newly integrated one: visual servoing.

In the QP control framework, each task $\boldsymbol{\mathcal{T}_i}$ is defined as an error in the sensory task space; this error is a function of the QP decision variables that are: (i) robot configuration accelerations, $\boldsymbol{\ddot{q}}$, (ii) the actuator torques, $\boldsymbol{\tau}$, and (iii) the contact forces, $\boldsymbol{f}$. To the task, we associate a Jacobian matrix $\boldsymbol{J_i}$, a weight $\boldsymbol{w_i}$ that defines the soft priority, and gains (namely, a stiffness $\boldsymbol{k_i}$).

Let us refer to Figure 2.1 for the illustrated example of the HRP-4 humanoid robot, controlled by the QP, performing a complex task. At the high level, the given mission can be formulated as: "operating the circuit breaker". At the low level, such a mission is split into a set of tasks that can be written as constraints or as objectives on the decision variables. Hence, tasks $\boldsymbol{\mathcal{T}_i}$ are formulated as errors in the robot sensory space and used in the QP as constraints, or as part of the objective function with appropriate values for stiffness, $\boldsymbol{k_i}$, and weight, $\boldsymbol{w_i}$. All the QP tasks, $\boldsymbol{\mathcal{T}_i}$, which are necessary to perform one sample mission are exemplified and visually illustrated in Figure 2.1, along with the set of QP constraints, such as joint limits, contact friction cone constraints, etc. We bring more general mathematical formulation of the QP in the continuation of this section.

The most usual tasks, in the QP control framework, are position-based (an error between any point $\boldsymbol{p}_{\text{robot}}(\boldsymbol{q})$ of the robot, like an end-effector position, center of mass (CoM) position etc., and a desired position of this point, $\boldsymbol{p_{target}}$). Here, $\boldsymbol{q}$ is the robot generalized configuration variables, i.e. the vector containing the position and the orientation of the floating base, plus the joint values. For $\boldsymbol{N_p}$ number of such tasks with errors $\boldsymbol{\mathcal{T}_p} = \boldsymbol{p}_{\text{robot}}(\boldsymbol{q}) - \boldsymbol{p}_{\text{target}}$, weights $\boldsymbol{w_p}$, Jacobian matrices $\boldsymbol{J_p}$ and gains $\boldsymbol{k_p}$, we write the cost function:

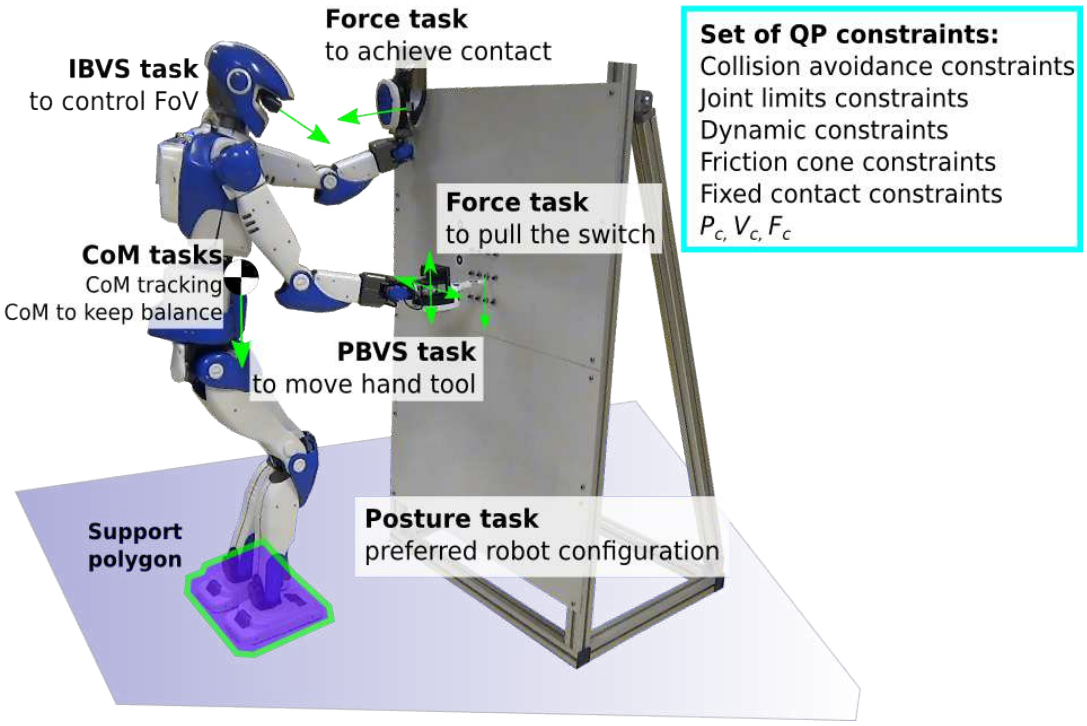


Figure 2.1: Whole-body control QP framework tasks (components of the QP objective function) and constraints illustrated on the example of the HRP-4 humanoid robot performing a complex task of operating a circuit breaker (see experiments in Chapter 4).

$$\mathcal{P} = \sum_{p=1}^{N_p} w_p \left\| \ddot{\mathcal{T}}_p + 2\sqrt{k_p}\dot{\mathcal{T}}_p + k_p \mathcal{T}_p \right\|^2 \tag{2.1}$$

with $\dot{\mathcal{T}}_p = J_p \dot{q}$, and $\ddot{\mathcal{T}}_p = J_p \ddot{q} + \dot{J}_p \dot{q}$. Tasks in position with strict inclusions, equalities and inequalities are integrated in the constraints part after two derivations that result in a linear form in the decision variables; they are gathered into $\mathcal{P}_c$.

In Figure 2.1, an example of tasks, which are summed up in $\mathcal{P}$ are: (i) one of the "**CoM Tasks**" is the CoM tracking tasks defined as an error between the current CoM and a desired one, CoM_d, that has a low priority, (ii) the "**Posture task**" is a tracking task with very low priority to have a preference posture/configuration of the robot, q_d, and to avoid singularities in the QP solver.

Position tasks belonging to $\mathcal{P}_c$ are:
(i) the kinematic (fixed) contact tasks. Once the contact is reached, it is shifted into the constraint part of the QP through the following task:

$$J\ddot{q} + \dot{J}\dot{q} = 0 \tag{2.2}$$

for example, each foot-ground contact is such kind of task; it writes as an equality

in the constraint part of the QP between the task-frame attached to the feet and a task frame attached to the ground;
(ii) another of the "**CoM Tasks**" ensures that the projection of the robot CoM lies within the region defined by the support (convex) polygon, and hence is written with a set of inequalities. This task is put in the constraint part to ensure that the robot does not lose balance and fall while operating a circuit breaker;
(iii) all collision avoidance tasks (not illustrated). Collision avoidance is also integrated into the QP as described in [47] with the following constraints:

$$\dot{\boldsymbol{d}} + \boldsymbol{t}\,\ddot{\boldsymbol{d}} \;\geq\; \boldsymbol{\xi}\frac{\boldsymbol{d} - \boldsymbol{d_\sigma}}{\boldsymbol{d_i} - \boldsymbol{d_\sigma}} \tag{2.3}$$

where $\boldsymbol{d}$ is the distance between a pair of bodies computed using [48], $\boldsymbol{d_\sigma}$ is the threshold distance, under which we consider that the collision happens; $\boldsymbol{d_i}$ is the influence distance to activate the damping; $\boldsymbol{\xi}$ is a damping coefficient and $\boldsymbol{t}$ is the control time-step. The distance is a function of the joint state, and its double derivative writes in terms of the configuration acceleration and in a linear form.
(iv) Joint limits tasks (not illustrated) that write simply as:

$$\boldsymbol{q_{\min}} \leq \boldsymbol{q} \leq \boldsymbol{q_{\max}} \tag{2.4}$$

Let $\boldsymbol{N_v}$ be the number of visual servoing tasks, which are defined as residuals in the task function space $\boldsymbol{\mathcal{T}_v} = \boldsymbol{s(q)} - \boldsymbol{s}_{\text{target}}$, $\boldsymbol{s(q)}$ being the visual features as currently observed by the robot in configuration $\boldsymbol{q}$, and $\boldsymbol{s}_{\text{target}}$ their desired value. The time derivative of the residual is related to the joint velocities through the Jacobian $\boldsymbol{J_v} = \boldsymbol{L_v J}$, $\boldsymbol{L_v}$ being the interaction matrix (i.e. the visual features Jacobian) and $\boldsymbol{J}$ being that of the robot as in classical visual servoing formulation. Two main types of visual servoing approaches and corresponding interaction matrices forms are discussed in more detail later in this chapter, in sections 2.3 and 2.4. Now we have that, $\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{v}} = \boldsymbol{L_v J}\dot{\boldsymbol{q}}$ and its derivative $\ddot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{v}} = \dot{\boldsymbol{L}}_{\boldsymbol{v}}\boldsymbol{J}\dot{\boldsymbol{q}} + \boldsymbol{L_v}\dot{\boldsymbol{J}}_{\boldsymbol{v}}\dot{\boldsymbol{q}} + \boldsymbol{L_v J}\ddot{\boldsymbol{q}}$ are integrated into the cost function with weight $\boldsymbol{w_v}$ and gain $\boldsymbol{k_v}$ as:

$$\boldsymbol{\mathcal{V}} = \sum_{\boldsymbol{v=1}}^{\boldsymbol{N_v}} \boldsymbol{w_v} \left\| \ddot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{v}} + \boldsymbol{2}\sqrt{\boldsymbol{k_v}}\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{v}} + \boldsymbol{k_v}\boldsymbol{\mathcal{T}_v} \right\| \tag{2.5}$$

More details on this formulation can be found in [38]. Here also, visual tasks with strict inclusion, equalities or inequalities are integrated into the constraints part as before and gathered in $\boldsymbol{\mathcal{V}_c}$.

In Figure 2.1, an example of tasks belonging to $\boldsymbol{\mathcal{V}}$ is the "**PBVS task**", or position based visual servoing task, that achieves visual servoing between the right arm tool to reach the exact position of the desired panel's button to pull. The "**PBVS task**" task is assigned relatively high weight to ensure precise and correct manipulation of the small buttons. Whereas another example of a task belonging to $\boldsymbol{\mathcal{V}_c}$, **"IBVS task"**, or image based visual servoing task, which is responsible

for keeping visual markers in the field of view (FoV), can be assigned a much lower weight value, to allow the FoV to deviate from its target value by a few centimetres as long as it does not affect the visibility of the markers. The detailed expression of the visual task is thoroughly developed in the coming sections of this chapter.

Force control is implemented as an interplay between two functions. First, the target force, $\boldsymbol{f}_{\text{target}}$, can be either user defined, $\boldsymbol{f_d}$, or it can be the output of the QP controller, $\boldsymbol{f}$. Let $\boldsymbol{N_f}$, be the number of force control tasks. Since the force $\boldsymbol{f}$ is a QP decision variable, then if $\boldsymbol{f_d}$ is defined, the corresponding force task writes simply as $\boldsymbol{\mathcal{T}_f} = \boldsymbol{f} - \boldsymbol{f_d}$. In this latter case, $\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} = \ddot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} = \boldsymbol{0}$ by definition; it is the "QP force task" in Figure 2.2. Whether $\boldsymbol{f_d}$ is defined or not, we also propose an admittance task exactly in a form of a position-type one (previously discussed), where the $\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} = \dot{\boldsymbol{p}}_{\text{robot}} - \dot{\boldsymbol{p}}_{\text{target}}$ such that $\dot{\boldsymbol{p}}_{\text{target}} = \boldsymbol{K_f}(\boldsymbol{f}_{\text{target}} - \boldsymbol{f}_{\text{sensor}})|_{\boldsymbol{n}}$. $\ddot{\boldsymbol{p}}_{\text{target}}$ and $\boldsymbol{p}_{\text{target}}$ are obtained by numerical derivation and integration respectively; $\boldsymbol{K_f}$ is a gain and $\boldsymbol{n}$ is the surface normal; this is the "QP admittance task" in Figure 2.2.

Finally, force tasks weighted error is:

$$\boldsymbol{\mathcal{F}} = \sum_{\boldsymbol{f=1}}^{\boldsymbol{N_f}} \boldsymbol{w_f} \left\| \ddot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} + \boldsymbol{2}\sqrt{\boldsymbol{k_f}}\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} + \boldsymbol{k_f}\boldsymbol{\mathcal{T}_f} \right\| \tag{2.6}$$

It is understood that Equation (2.6) can be one or duplicate (i.e. two equations). It is two equation when $\boldsymbol{f_d}$ is specified, the first one corresponds to the task $\boldsymbol{\mathcal{T}_f} = \boldsymbol{f} - \boldsymbol{f_d}$ with $\dot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} = \ddot{\boldsymbol{\mathcal{T}}}_{\boldsymbol{f}} = \boldsymbol{0}$, and the second one is the admittance task previously discussed. If in $\boldsymbol{f_d}$ is not specified, Equation (2.6) is the single admittance task, see Figure 2.2. As for previous task-types, strict inclusions, inequalities and equalities can be defined as constraints gathered in $\boldsymbol{\mathcal{F}_c}$.

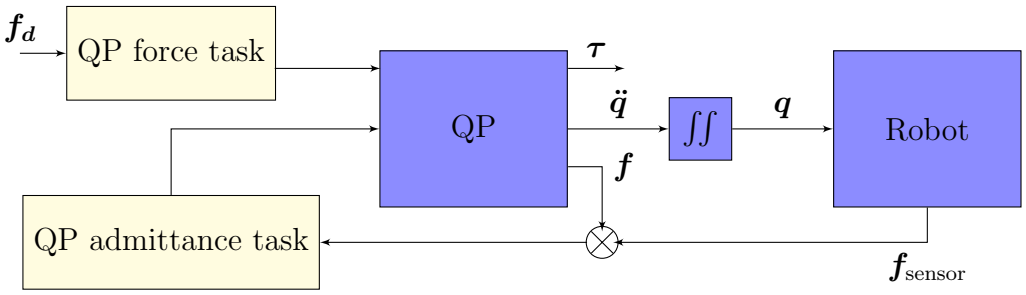


Figure 2.2: Implementation of the force regulation within the QP framework.

In Figure 2.1, an example of tasks belonging to $\boldsymbol{\mathcal{F}}$ is the **"Force task"** that achieves admittance control of the left arm taking a contact on the panel when previously reached by a contact reaching task that is not illustrated. Whereas examples of tasks belonging to $\boldsymbol{\mathcal{F}_c}$ are:
(i) the friction cone constraints (not illustrated):

$$\boldsymbol{Af} \leq \boldsymbol{0}, \tag{2.7}$$

for non-sliding contacts. This task keeps the forces $\boldsymbol{f}$ within their linearized friction cones represented by $\boldsymbol{A}$ (see [49]).
(ii) a threshold of contact forces, e.g. an inequality threshold put on the contact force measured from the left wrist embedded force/torques transducer (i.e. 6D force sensor):

$$\boldsymbol{f_n} \leq \boldsymbol{f}_{\text{threshold}} \tag{2.8}$$

At each control time step $\boldsymbol{t}$, the controller with the set of tasks, previously defined, is fed back with the current state of the robot $\boldsymbol{(q, \dot{q})}$ and the sensor parameters. It then solves for the decision variables: robot state acceleration $\ddot{\boldsymbol{q}}$, the stacked vector of forces $\boldsymbol{f}$, and the actuation torques $\boldsymbol{\tau}$ through the following QP:

$$\min_{(\ddot{\boldsymbol{q}}, \boldsymbol{f}, \boldsymbol{\tau})} \boldsymbol{\mathcal{P}} + \boldsymbol{\mathcal{V}} + \boldsymbol{\mathcal{F}} \tag{2.9}$$

subject to: $\boldsymbol{\mathcal{P}_c}$, $\boldsymbol{\mathcal{V}_c}$, $\boldsymbol{\mathcal{F}_c}$, additionally to the common constraints:

$$\boldsymbol{M(q)\ddot{q} + C(q, \dot{q}) + G(q) = S\tau + J^T f} \tag{2.10}$$

which is the dynamic equation linking all the decision variables, with $\boldsymbol{S}$ a selection matrix for the actuated joints in $\boldsymbol{q}$, $\boldsymbol{J}$ is the force to torques mapping Jacobian. $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$ are the classical Inertia matrix, the Coriolis and Gravitation vectors respectively. By pre-multiplying this equation by $\boldsymbol{S^T}$, we can express $\boldsymbol{\tau}$ as an affine function of $\ddot{\boldsymbol{q}}$ and $\boldsymbol{f}$, and remove it from the decision variables.

The torque limits task inherent from the actuators' characteristics are expressed as:

$$\boldsymbol{-\tau_{max} \leq \tau \leq \tau_{max}} \tag{2.11}$$

Thanks to the torque constraints, the torque decision variable can be eliminated from the QP decision variable ending up having a QP only in terms of the configuration acceleration $\ddot{\boldsymbol{q}}$ and force variables $\boldsymbol{f}$. That is:

$$\boldsymbol{-S\tau_{max} \leq M(q)\ddot{q} + C(q, \dot{q}) + G(q) - J^T f \leq S\tau_{max}} \tag{2.12}$$

As in [47], waypoint tasks are integrated as guide-paths to avoid local minima. They also resolve the contradiction between moving a body to a desired contact spot and at the same time avoiding collision between that body and the environment component on which the contact is defined. Note that our controller can integrate multiple robots in a single formulation (i.e. any objects, including other robots, that can be represented as robotic structures even with passive joints), see [50] and Chapter 4 for more details.

In the continuation of this chapter, we will focus in more detail on the visual servoing tasks.

2.3 Image Based Visual Servoing task

The aim of a visual servoing task is to change the configuration of the system in order to minimize the error between values of desired and observed features extracted from the vision. In the image based visual servoing (IBVS) the error is defined as a difference between currently observed ($\boldsymbol{s}$) and desired ($\boldsymbol{s^*}$) features in a 2D image plane. Commonly used features are points, lines or other geometrical shapes detected in the image. For simplicity of explanation, we consider a point feature with coordinates in the image plane $\boldsymbol{(u, v)}$, expressed in pixels. Suppose the desired location of this point feature is $\boldsymbol{(u^*, v^*)}$ (the center of the image). First the error is computed:

$$\boldsymbol{e = s - s^* = (u, v) - (u^*, v^*)} \tag{2.13}$$

Now, the mapping $\boldsymbol{L}$ between camera spatial velocity, $\boldsymbol{v_c}$, and point feature velocity, $\boldsymbol{\dot{s}}$, can be derived. Since $\boldsymbol{s^*}$ is constant for the visual servoing task, derivative of the point feature is equal to the error derivative:

$$\boldsymbol{\dot{s} = \dot{e} = L v_c} \tag{2.14}$$

In the motion control literature, mapping $\boldsymbol{L}$ is called either **interaction matrix** or **image Jacobian**. For many common image features, the derivation of the $\boldsymbol{L}$ mapping form is performed using feature geometry [36]. For instance, $\boldsymbol{L}$ mapping for a point feature is the following $\mathbf{2 \times 6}$ matrix:

$$\boldsymbol{L_p} = \begin{bmatrix} -\frac{\mathbf{1}}{\boldsymbol{Z}} & \mathbf{0} & \frac{\boldsymbol{u}}{\boldsymbol{Z}} & \boldsymbol{uv} & -(\mathbf{1} + \boldsymbol{u^2}) & \boldsymbol{v} \\ \mathbf{0} & -\frac{\mathbf{1}}{\boldsymbol{Z}} & \frac{\boldsymbol{v}}{\boldsymbol{Z}} & \mathbf{1} + \boldsymbol{v^2} & -\boldsymbol{uv} & -\boldsymbol{v} \end{bmatrix} \tag{2.15}$$

where $\boldsymbol{Z}$ is the estimated depth of the point $\boldsymbol{(u, v)}$ relative to the camera frame. For more complex features the interaction matrix can be approximated or learned offline.

In order to enable an exponential decrease of the error, the following relation is specified with $\boldsymbol{\lambda > 0}$:

$$\boldsymbol{\dot{e} = -\lambda e} \tag{2.16}$$

As a result of relations 2.14 and 2.16, the velocity of the camera can be computed as follows:

$$\boldsymbol{v_c = -\lambda L^+ e} \tag{2.17}$$

where $\boldsymbol{L^+}$ denotes the pseudoinverse of the interaction matrix $\boldsymbol{L}$. The value of the $\boldsymbol{v_c}$ is passed as an input to the motion update, and the corresponding motion is performed, the process of measuring the new position of the point feature, recomputing the error and the interaction matrix and finally obtaining the new value of the camera velocity can be repeated iteratively until the error converges to zero.

2.4 Position Based Visual Servoing task

In the case of position based visual servoing (PBVS) the error is defined in terms of difference in rotation and/or translation of two reference frames in the 3D space. First of all, the features are detected in the image plane. Then, the 3D position and orientation of those features need to be estimated. This can be done, for example, by forward projecting the points from the image plane using the camera calibration parameters. Once the pose is estimated, the motion update can be defined so that the robot end-effector's reference frame origin is aligned with the pose of the point feature. Such computations are useful for object manipulation.

Suppose that the goal is to position the right hand of the robot to be aligned with a given point $\boldsymbol{P}$ in the 3D space detected by the camera. In terms of the PBVS task, this can be achieved by minimizing the error between $\boldsymbol{P}$ and the origin of the right hand link reference frame, $\boldsymbol{T}_{\text{rh}}$). An important thing to notice here is that both points are expressed in the camera reference frame. The feature in PBVS can be of the following type:

$$\boldsymbol{s} = (\boldsymbol{t}, \boldsymbol{\theta u}), \tag{2.18}$$

here $\boldsymbol{t}$ is the translation difference between frames $\boldsymbol{P}$ and $\boldsymbol{T}_{\text{rh}}$ and $\boldsymbol{\theta u}$ is the rotation difference between two frames expressed in axis angle coordinates. In order to align two frames, both $\boldsymbol{t}$ and $\boldsymbol{\theta u}$ need to be 0, which means that $\boldsymbol{s^*} = \boldsymbol{0}$ and $\boldsymbol{e} = \boldsymbol{s}$.

For such a definition of the PBVS task, the interaction matrix takes the following form:

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{L_{\theta u}} \end{bmatrix}, \tag{2.19}$$

where R is a rotation matrix between frames $\boldsymbol{P}$ and $\boldsymbol{T}_{\text{rh}}$ and

$$\boldsymbol{L_{\theta u}} = \boldsymbol{I_3} - \frac{\boldsymbol{\theta}}{\boldsymbol{2}}[\boldsymbol{u}]_{\boldsymbol{x}} + \left(\boldsymbol{1} - \frac{\boldsymbol{\operatorname{sinc}\theta}}{\boldsymbol{\operatorname{sinc}^2}\frac{\boldsymbol{\theta}}{\boldsymbol{2}}}\right)[\boldsymbol{u}]_{\boldsymbol{x}}^{\boldsymbol{2}}, \tag{2.20}$$

where $\boldsymbol{\operatorname{sinc}\theta}$ is the sinus cardinal defined such that $\boldsymbol{\theta \operatorname{sinc}\theta} = \boldsymbol{\sin\theta}$ and $\boldsymbol{\operatorname{sinc}(0) = 1}$. The interested reader can refer to a tutorial on VS [36] for the derivation of the interaction matrix form.

The mapping $\boldsymbol{L}$ from 2.19 relates the $\boldsymbol{T}_{\text{rh}}$ frame velocity, $\boldsymbol{v}_{\boldsymbol{T}_{\text{rh}}}$, to the feature error velocity as follows:

$$\dot{\boldsymbol{e}} = \boldsymbol{L}\boldsymbol{v}_{\boldsymbol{T}_{\text{rh}}} \tag{2.21}$$

The computation of the $\boldsymbol{v}_{\boldsymbol{T}_{\text{rh}}}$ frame velocity is now analogous to the computation of the camera velocity in equation 2.17.

2.5 Sample visual servoing experiment: gazing

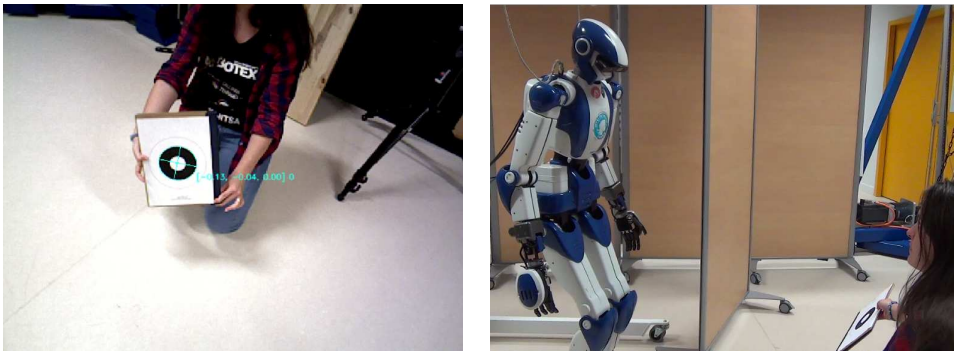
We have assessed the QP integration of the visual servoing task with preliminary experiments on the HRP-4 consisting in gazing and whole-body tracking of a marker (Figure 2.3).

For a gazing experiment, a WhyCon marker [51] was used as a visual target. The robot was required to minimize the error between the detected position of the marker in the image plane and the center of the image. In order to achieve that, an IBVS task was defined and added to the QP with the following task error:

$$\boldsymbol{e} = (\boldsymbol{u}_{\text{wc}}, \boldsymbol{v}_{\text{wc}}) - (\mathbf{0}, \mathbf{0}), \tag{2.22}$$

where $\boldsymbol{u}_{\text{wc}}$ and $\boldsymbol{v}_{\text{wc}}$ are the pixel coordinates of the detected WhyCon marker (Figure 2.3a).

Additionally, posture and CoM tasks were added to the same QP instance to ensure that the posture of the robot remains as preferred while the gazing task is performed and that the robot does not fall. IBVS, posture and CoM tasks were assigned weights 50, 10 and 10000 respectively for the optimisation, meaning that the IBVS task had a higher priority than the posture task, but much lower than the CoM task, to ensure that the robot could move while performing the IBVS task without falling.



(a) Robot point of view (PoV). (b) View from the side.

Figure 2.3: IBVS experiment.

2.6 QP controller implementation: manipulation with active perception

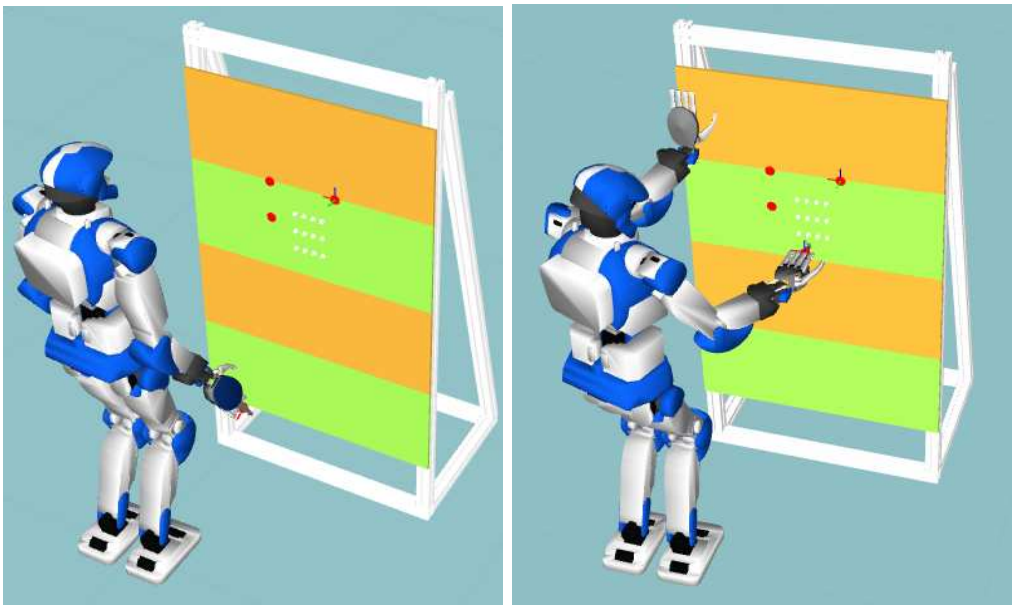
For more complex experiments, IBVS and PBVS tasks can be combined in a single QP instance. While a PBVS task is used for manipulation, the IBVS task can be used to update the pose of the camera so that the manipulated object is ensured to be in the FoV during the entire process of manipulation.

In order to study visual servoing integration in the QP control framework, a sample case of the HRP-4 humanoid robot operating an Airbus circuit breaker was implemented in the form of a QP controller including position, posture, force and other classical QP framework tasks, along with IBVS and PBVS visual servoing tasks, which allowed us to enable closed-loop control based on visual feedback. In this section, we bring the implementation and simulation details of the controller and discuss experimental results of a real setup in Chapter 4.

In the experimental setup for circuit breaker operation, four markers were used. One of the markers was attached to the robot hand tool, and the remaining three were attached to the wall in the environment in front of the robot. The robot was required to pull the switches/buttons on the wall with the right hand tool. In order to enable this manipulation, the PBVS task was defined and added to the QP with the corresponding error, $\boldsymbol{e} = (\boldsymbol{t}, \boldsymbol{\theta u})$, where $\boldsymbol{t}$ is a difference in translation and $\boldsymbol{\theta u}$ is a difference in rotation of the right hand tool marker frame and fixed manipulated button reference frame. Both frames are expressed in the camera coordinate frame. This is illustrated in the sample frame of simulated FoV of the robot in Figure 2.5. The manipulated button reference frame position and orientation are obtained by adding the offsets in $\boldsymbol{X}$ and $\boldsymbol{Y}$ axes of the wall reference frame, which is computed from position of 3 markers attached to the wall. In the described experiment only the position of the right hand frame was controlled by the PBVS task, while the orientation task was defined separately with a fixed hand orientation goal.

In order to ensure that both markers are in the camera FoV, an IBVS task is defined as described in Subsection 2.5, to minimize the error between the image coordinates of the right hand tool frame and the image center. The IBVS task is added to the QP after the right hand end-effector has been positioned in front of the robot somewhat close to the manipulated switches. Three markers are positioned on the wall relatively close to the switches so that during the manipulation experiment they remains in the FoV as the hand tool is being tracked with the camera. The frames of the simulation at different parts of experiment are shown in Figure 2.4.

We also exploit task-aware contact planing in the implementation of the controller, by creating a closed-kinematic chain between the wall and robot body. This is achieved by maintaining a contact between the robot's left hand and the wall (as shown in Figure 2.4b). Creating such a closed-kinematic chain increases the robot's equilibrium and pulling forces. Contact is implemented as a force and admittance tasks in the QP, resulting in the integrated force and vision based controller. The addition and removal of the QP tasks and constraints in the controller is managed by a finite state machine (FSM) composed by a finite set of states and transitions. More details and the real experimental result of such an integration in the context of the Airbus circuit breaker operating use-case are presented in Chapter 4.



(a) Start of the simulation with only posture, CoM and torso orientation tasks in the QP cost function.

(b) Simulation state after the IBVS, PBVS and left hand force tasks are added to the QP cost function.

Figure 2.4: Experimental setup shown in simulation.

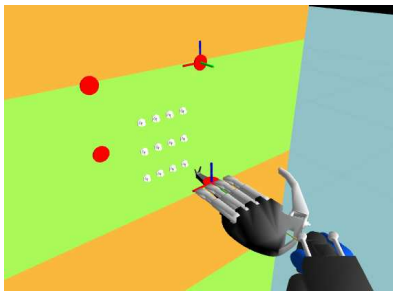


Figure 2.5: Camera view after IBVS and PBVS tasks are added to the QP cost function. All markers are in the FoV, wall and tool marker frames are visible.

2.7 Conclusion

We have presented the QP control framework formulation and explained the working principles of vision based motion control. We discussed a sample controller implementation, which uses VS tasks to enable the HPR-4 to perform complex closed-loop manipulation. Real experiments are discussed later in Chapter 4.

In fact, now that we have tools for VS, we will use them in order to estimate the configuration of the articulated objects that are manipulated by the robot. The idea is to track continuously the configuration of these objects (recall that they are modelled as "robots") using the so-called Virtual Visual Servoing, which is explained in detail in the following Chapter 3.

Chapter 3

Articulated object configuration estimation via visual tracking

In the previous chapter, we described the whole-body motion control in the QP framework, the vision-based control and visual servoing. In this chapter, we describe methodology of using point and edge features in combination with visual servoing principles to enable estimation of poly-articulated object configuration. We describe the solution for real-time and accurate markerless edge feature extraction that can be used as a feedback in the closed-loop control. The proposed methodology is exploited later in Chapter 4, in order to effectively close the loop on perception in the multi-robot QP (MQP) control framework, and perform real closed-loop experiments.

3.1 Introduction

The QP control framework, presented in Section 2.2, can be used for object manipulation and interaction. This is achieved by modelling the manipulated object as a "robot" with its corresponding configuration. The configuration of the object consists of the floating base (FB) position and orientation and, for poly-articulated objects, joint values. The extension of the QP control framework to multi-robot control is explained in Section 4.1.

Since daily objects are not equipped with encoders or other sensors, their configuration needs to be estimated at each time-step. We use visual tracking techniques to achieve the block **M** in Figure. 1 in real-time. Since we do not track a target but instead estimate the configuration by continuously matching the image plane configuration on the 3D model, such an approach is called Virtual Visual Sevoing. We bring formulation of the method in the following sections of this chapter.

3.2 Estimation of the configuration using visual tracking: background

Reconstruction of the configuration of articulated objects is a well studied problem in both the computer vision community (tracking human motion, hands, etc.), and in robotics (tracking robotic systems). In this section we look at the related works purely from the point-of-view of the configuration reconstruction problem and not from the visual perception viewpoint, which was done in Chapter 1.

Several methods to track motion of articulated bodies have been proposed over the years. A kinematic tree based parametrization of articulated objects was used in [33]. In the latter work, the tracking was formulated as a tree parameters fitting problem, assuming full geometric model of an object to be known. In [32] the articulated structure tracking is handled as an extension of rigid object tracking. Independent trackers are used to compute motions of every link, then constraints between the links, expressed using Lie algebra formalism, are imposed to find optimal set of motions that satisfy the constraints. Multiple hypothesis using derivative- or gradient-free optimization techniques have been studied in the line of works (e.g. [15][16][17], also described in Chapter 1). Such approaches are quite helpful to avoid local minima in optimization. As an example of application, such methods have been recently studied further for use in force estimation from vision [18][19]. Yet, estimating the configuration of articulated structures with multiple hypothesis methods, can lack frame-to-frame consistency, due to occasional ambiguous observations which cause false hypotheses temporarily to have high matching scores. This issue is not critical in many applications areas (surveillance, computer graphics, etc.), but in a robotic closed-loop control schemes, such estimation inconsistency could result in a very bad behavior, such as jumps and jerky motions.

As mentioned previously in Subsection 1.4.2, the motion of articulated objects can also be estimated using depth information in a GPU-based implementation of an Extended Kalman Filter (EKF) in [29]. This method has been extended further to consider physical constraints in tracker objective function by using contact information in robotic object manipulation scenario [52]. Another depth based method for estimating the state of a robotic arm was proposed in [53]. It demonstrated robustness to calibration errors in a closed-loop manipulation task. Combination of depth and joint encoders data was used in [54] to track the state of robotic arm. However, for some robotic platforms, especially humanoids, end-effector distance from the camera may not exceed the minimum distance for depth data acquisition by a standard range sensor. In such case, RGB data processing is the only reliable source of visual information. Articulation tracking can also be done using images collected by a multi-camera system and then processed by a particle filter [55]. These methods are yet computationally expensive to be applied at the low-level real-time robot control.

Model-based approaches that make use of monocular images result to be the most promising techniques for achieving fast and accurate estimate of articulated objects configuration. In [56], a Kalman filter based tracking, using multiple models (such as the geometric and the appearance models of the object), was proposed to recover values of joint position and velocity, but not that of the floating base. Another articulated object tracking method has been proposed in [34]. This algorithm uses a virtual visual servoing approach [35][27], previously mentioned in Subsection 1.4.1, to minimize the visual error between some detected features of the real object and other virtually projected by the estimated system. The *virtual* control law provides the configuration of the object, but not expressed with the classical generalized coordinates (see Equation 1.3 in Subsection 1.4.4 for detail). Further computations should be added to retrieve the joint velocity variable values.

An advantage of the model-based approaches is that the object and the visual features trackers can work together cooperatively: knowing the model of the object, the tracking of features leads to the reconstruction of the object configuration and *vice-versa*. Extending this concept, the features motion can also reveal geometric information of the observed object, that in turn is used to better track the features. This idea is exploited in [57], that estimates the kinematic structure of the observed object combining the manipulation task with the perception algorithm. With the same principle, color and depth (RGB-D) information are processed by an EKF to provide also a measurement of the joint values in [58]. In [59], RGB-D data was processed in a unified framework able to estimate the pose, the shape and the structure of the observed object. These approaches do not need the model of the object, being itself estimated, but have been validated with simple articulated objects, they are not guaranteed to converge fast or to be reliable in all circumstances. Some are computationally expensive.

In our attempt to find the articulated motion tracking solution, whose formalism can be used as a part of MQP in the closed-loop object manipulation control, no existing method could fit our requirements. Therefore, we took inspiration from previous works to devise a tracker whose formalism suits the MQP requirements. We present our methodology in the remaining sections of this chapter.

3.3 Virtual Visual Servoing for articulated object configuration estimation

To track articulated objects for robotic manipulation, we propose a method based on virtual visual servoing. We briefly recall the basics of this technique to formulate our tracking problem.

Besides vision based motion control, visual servoing (VS), introduced in the previous chapter, can also serve a purpose of object configuration estimation. This

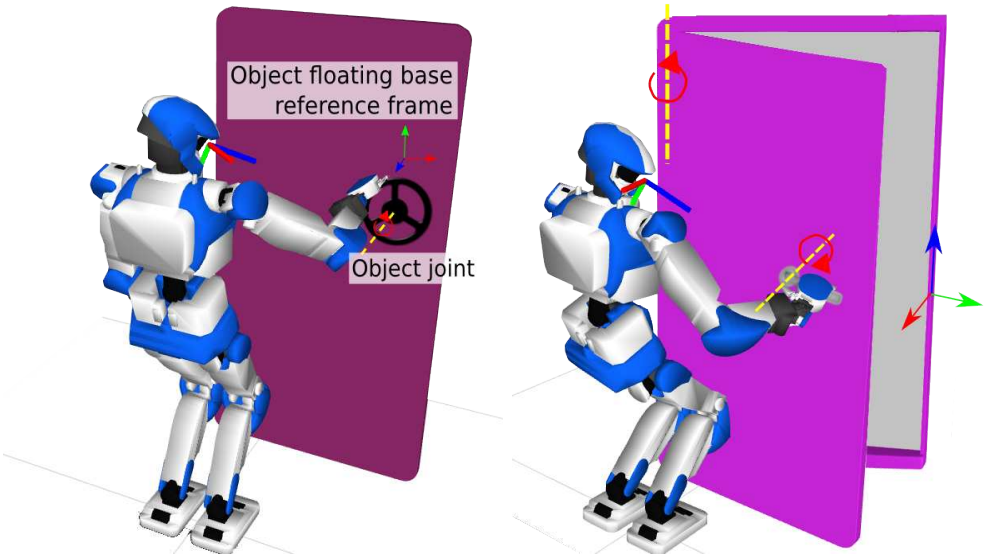
is done by the means of the Virtual Visual Servoing (VVS) paradigm, initially introduced in [35] for the inclusion of computer graphics generated object into off-line or real-time recorded videos: this is called augmented-reality. In this latter context, it is important to well track the real environment in order to well position the virtual (i.e. graphic) objects in the scene. Recently, this is done thanks to simultaneous localization and mapping (SLAM) technologies. In our work, we study the use of VVS to estimate the configuration of the poly-articulated object in a form required by Multi-robot QP control framework for further use in object manipulation. The integration is explained in Chapter 4. Following in this section, we rather look into more details on the VVS idea and give a formulation of VVS based poly-articulated object configuration estimation.

We recall the general idea of VVS. For simplicity, let's first consider the case of a rigid and non poly-articulated object configuration. Suppose we have $\boldsymbol{m}$ visual features on the object; each feature $\boldsymbol{m_i}$ is of dimension $\boldsymbol{d_i}$ and hence $\mathbf{dim}(\boldsymbol{m}) = \boldsymbol{d_m} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{d_i}$. The visual features can be observed by the camera, detected and recognized through real-time image processing. The configuration of the object, $\boldsymbol{q}$, w.r.t. the camera frame and part of the corresponding features positions on the image $\boldsymbol{s(q)}$ are known at time $\boldsymbol{t}$. When a new camera frame is acquired, new position of the visual features, $\boldsymbol{s^*}$, can be extracted by means of computer vision. The new position of the visual features, measured at time $\boldsymbol{t+1}$, is compared with the position of the same features at time $\boldsymbol{t}$. The corresponding errors are computed and stored in the vector $\boldsymbol{e = s(q) - s^*}$. We know, from the classical visual servoing techniques, that the error vector can be related to the camera velocity through the interaction matrix mapping, $\boldsymbol{L}$. Then for all features tracked, we stack all corresponding interaction matrices. Since the camera, in this case, is a central reference frame, and the new object configuration, $\boldsymbol{q^*}$, only needs to be estimated w.r.t. the camera, knowledge on the camera velocity is sufficient to derive the new configuration of the object in the camera reference frame.

Now, we explain how the VVS based method can be used for poly-articulated object configuration reconstruction. Suppose a humanoid robot is manipulating a poly-articulated object with a camera looking at the object, as in the depicted scenarios of Figure 3.1; more specifically, let us consider the case where a humanoid robot is opening the drawer of a furniture (Figure 3.1c), as this scenario is used later for real experiments.

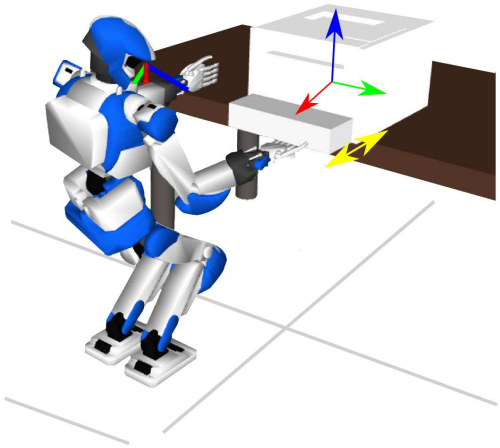
The camera frame of the robot, $\boldsymbol{\mathcal{F}_c}$, is used as the central reference, meaning that reconstructed configuration of the object is expressed in $\boldsymbol{\mathcal{F}_c}$. Figure 3.1 demonstrates several examples of $\boldsymbol{\mathcal{F}_c}$ axes placement. Another frame of interest, $\boldsymbol{\mathcal{F}_o}$, is arbitrarily defined at the floating base (FB) of the object, also shown in Figure 3.1. The configuration of the poly-articulated object, which needs to be estimated, is denoted by the vector

$$\boldsymbol{q^*} = \begin{pmatrix} \boldsymbol{p_o} \\ \boldsymbol{\sigma_o} \\ \boldsymbol{q_{joint}} \end{pmatrix} \in \mathbb{R}^{\boldsymbol{7+n}} \tag{3.1}$$



(a) HRP-4 manipulating a steering wheel (revolute or helical joint).

(b) HRP-4 opening a door (revolute joints: the handle and the door).



(c) HRP-4 manipulating the drawer of a furniture (prismatic joint).

Figure 3.1: Manipulation of various articulated objets by a humanoid robot. The embedded camera reference frame, $\boldsymbol{\mathcal{F}_c}$, is indicated in front of the robot's 'head'.

where $\boldsymbol{p_o} \in \mathbb{R}^3$ and $\boldsymbol{\sigma_o} \in \mathbb{R}^4$ are the position and orientation (expressed with a quaternion) of $\boldsymbol{\mathcal{F}_o}$ w.r.t. $\boldsymbol{\mathcal{F}_c}$. Quaternions prevent the orientation representation from being singular. The vector $\boldsymbol{q_{joint}} \in \mathbb{R}^n$ in (3.1) is the vector of the generalized

coordinates describing the internal configuration of the object, composed of $\boldsymbol{n}$ joints (e.g amount of drawer opening defined by the value of drawer prismatic joint in Figure 3.1c). We use a visual controller to estimate the derivative of the configuration vector. Thus, the estimate of $\boldsymbol{q}^*$ is reconstructed through numerical integration. Suppose we have an object with $\boldsymbol{p}$ known visual features. The error of the proposed control scheme is the difference between the previous known position of the visual features, $\boldsymbol{s(q)}$, where $\boldsymbol{q}$ is the previous known configuration vector value, and the current measured position of the corresponding visual features $\boldsymbol{s}^*$:

$$\boldsymbol{e} = \boldsymbol{s(q)} - \boldsymbol{s}^* \tag{3.2}$$

For the sake of generality, we do not specify which type of the visual features are used at this stage. In any case, the dynamics of this error says how the visual features changed between two time steps or between two sequentially acquired camera frames. Error dynamics can be written as follows:

$$\dot{\boldsymbol{e}} = \dot{\boldsymbol{s}}(\boldsymbol{q}) - \dot{\boldsymbol{s}}^* \tag{3.3}$$

The observable motion of the visual features is a consequence of the motion and corresponding change of the configuration of the object w.r.t. the camera. In $\boldsymbol{\mathcal{F}_c}$, the velocity of each visual feature can be expressed using the geometry and kinematics of the object ($\boldsymbol{i = 1, ..., p}$):

$$\boldsymbol{v_i} = \boldsymbol{J_i(q)\dot{q}} \tag{3.4}$$

where $\boldsymbol{J_i}$ is the $\boldsymbol{6 \times (7 + n)}$ Jacobian of $\boldsymbol{i}^{\text{th}}$ visual feature. Thus, we can express the dynamics of each visual feature as follows ($\boldsymbol{i = 1, ..., p}$):

$$\dot{\boldsymbol{s}}_{\boldsymbol{i}}(\boldsymbol{q}) = -\boldsymbol{L_i J_i(q)\dot{q}} = \boldsymbol{A_i\dot{q}} \tag{3.5}$$

where $\boldsymbol{L_i}$ is the interaction matrix (or the image Jacobian) associated to the $\boldsymbol{i}^{\text{th}}$ visual feature; it depends on the estimated depth of the corresponding feature, that is available in the estimation process.

Usually, the interaction matrix relates the 2D velocity of the visual features to the 6D velocity of the camera. If the camera is considered fixed[1], the apparent motion of the features is due to the motion of the observed object (actually, its features) in the opposite direction, that explains the minus in (3.5). Now, the dynamics of the error can be explicitly written as follows:

$$\dot{\boldsymbol{e}} = \boldsymbol{A\dot{q}} - \dot{\boldsymbol{s}}^* \tag{3.6}$$

where we define, following the nomenclature in [34], $\boldsymbol{A} = \boldsymbol{(A_1, \ldots, A_p)^T}$ as the $\boldsymbol{d_m \times (7 + n)}$ *articulation matrix*, that relates the FB and joints velocities of

[1] $\boldsymbol{\mathcal{F}_c}$ being the reference frame, the relative object-camera motion is actually meant as motion of the object w.r.t. the "fixed" camera. In practice, it might happen that the camera itself moves.

the articulated object to the velocity of the visual features. Imposing a stable dynamics of the error ($\dot{\boldsymbol{e}} = -\boldsymbol{\lambda e}$, $\boldsymbol{\lambda} > \mathbf{0}$), we derive the following "control law":

$$\dot{\boldsymbol{q}} = -\boldsymbol{\lambda}\underline{\mathrm{A}}^{+}\boldsymbol{e} + \underline{\mathrm{A}}^{+}\underline{\dot{\mathrm{s}}}^{*} \tag{3.7}$$

where $\boldsymbol{A}^{+}$ is a pseudoinverse of the articulation matrix $\boldsymbol{A}$ and the term depending on $\dot{\boldsymbol{s}}^{*}$ introduces a predictive term in order to improve the performance of the tracker as it anticipates on the motion and cancels related static errors. The bars below the variables denote that approximation is used. The local stability of the controller is ensured if the articulation matrix and its approximation are full-rank, and we use a good approximation of $\boldsymbol{A}$ and of the visual feature derivative.

At steady state, (3.7) produces an estimate of object's configuration (that is the FB and joint velocities) $\dot{\boldsymbol{q}}_{\boldsymbol{k}}$ at each discrete time-period $\boldsymbol{kT_s}$ ($\boldsymbol{T_s}$ being the sampling time of the algorithm and $\boldsymbol{k}$ the loop iterator), from which $\boldsymbol{q_k}$, an estimate of the object configuration, is obtained by numerical integration.

To avoid the error introduced by brute-force normalization-based methods, the derivative of the quaternion is integrated using a closed-form exponential map method [60]. The other elements of $\boldsymbol{q_k}$ are obtained with simple explicit Euler integration:

$$\boldsymbol{q_k} = \begin{pmatrix} \boldsymbol{p_{o,k-1}} + (\dot{\boldsymbol{p}}_{\boldsymbol{o,k}} + \dot{\boldsymbol{p}}_{\boldsymbol{o,k-1}})\frac{\boldsymbol{T_s}}{\mathbf{2}} \\ \exp(\boldsymbol{\Omega_{k-1}}\,\boldsymbol{T_s})\boldsymbol{\sigma_{o,k-1}} \\ \boldsymbol{q_{j,k-1}} + (\dot{\boldsymbol{q}}_{\boldsymbol{j,k}} + \dot{\boldsymbol{q}}_{\boldsymbol{j,k-1}})\frac{\boldsymbol{T_s}}{\mathbf{2}} \end{pmatrix} \tag{3.8}$$

where $\boldsymbol{\Omega}$ is the $\mathbf{4} \times \mathbf{4}$ skew matrix of the FB angular velocity.

Pseudoalgorithm, presented in Algorithm 1, shows the complete procedure for computing the vector $\boldsymbol{q}$.

The controller (3.7) has a threefold utility: (i) it works as visual features tracker, (ii) FB pose estimator, and (iii) object's joints estimator. From these two latter information, the pose of any link of the object can be reconstructed. In principle, in order to reconstruct the same information, one could use multiple instances of VVS for estimating the rigid body pose of each object link separately. The advantage of using (3.7) stands in managing all information in the same framework, activating a virtuous circle. Indeed, the visual features trackers helps the estimation of the object FB pose, which helps the tracking of all the links and the estimation of the object joints, that in turn helps to correct estimation of the visual features.

The method can reconstruct also other information about the object. In fact, it is also possible to estimate some geometric parameters, such as the length of the links or even the composition of the CAD model, that is assumed to be known. This can be achieved as an extension of our algorithm, by modelling the distance between visual features on the same link as *virtual* prismatic joint. This characteristic will be tested in real experiments and described in Chapter 4, Section 4.4.

Algorithm 1: VVS-based tracking of articulated objects. Each acquired image is processed to detect the 2D visual features that are collected in the vector $\boldsymbol{s}^*$. The position of the visual features on the 3D object model, $\boldsymbol{\rho}$, is computed using the estimate of the object configuration from the previous iteration, $\boldsymbol{q}$. For each visual feature with Jacobian ($\boldsymbol{J_i}$), the visual feature locations on the image plane according to $\boldsymbol{q}$ ($\boldsymbol{s_i(q)}$), interaction matrix ($\boldsymbol{L_i}$) and corresponding part of the articulation matrix ($\boldsymbol{A_i}$) and corresponding visual feature error ($\boldsymbol{e_i}$). Once Articulation matrix and error vector is fully constructed, $\dot{\boldsymbol{q}}$ is computed and $\boldsymbol{q}$ is obtained through numerical integration.

foreach *new image frame* $\boldsymbol{I}$ **do**
  $\boldsymbol{s}^* \leftarrow$ DETECT FEATURES($\boldsymbol{I}$) ;
  $\boldsymbol{\rho} \leftarrow$ UPDATE MODEL($\boldsymbol{q}$) ;
  $\boldsymbol{e} \leftarrow$ allocate space for error vector $\boldsymbol{e}$;
  $\boldsymbol{A} \leftarrow$ allocate space for the articulation matrix;
  foreach *visual feature* $\boldsymbol{i}$ **do**
    $\boldsymbol{J_i} \leftarrow$ COMPUTE JACOBIAN($\boldsymbol{q}, \boldsymbol{\rho_i}$) ;
    $\boldsymbol{s_i(q)} \leftarrow$ PROJECT($\boldsymbol{\rho_i}$) ;
    $\boldsymbol{L_i} \leftarrow$ COMPUTE INTERACTION MATRIX($\boldsymbol{\rho_i}, \boldsymbol{s_i(q)}$) ;
    $\boldsymbol{A_i} = -\boldsymbol{L_i}\,\boldsymbol{J_i}$;
    $\boldsymbol{e_i} = \boldsymbol{s_i(q)} - \boldsymbol{s_i^*}$;
  end
  $\dot{\boldsymbol{q}} = -\boldsymbol{\lambda A^+ e} + \boldsymbol{A^+ \dot{s}^*}$;
  $\boldsymbol{q} \leftarrow$ NUMERICAL INTEGRATION($\dot{\boldsymbol{q}}$) ;
end

3.4 Point feature based parameter estimation

One of the most simple and well studied types of visual features in visual servoing are point features. Let us define a vector $\boldsymbol{\rho} \in \mathbb{R}^{\boldsymbol{3p}}$ of $\boldsymbol{p}$ points of interest (PoI) on the articulated object, whose projection on the image plane of the camera provides $\boldsymbol{p}$ corresponding visual 2D point features.

The motion of the articulated object, $\boldsymbol{q}^*$, which we wish to estimate, produces the motion of visual features collected in the vector $\boldsymbol{s}^* \in \mathbb{R}^{\boldsymbol{2p}}$. These features are observable and measurable on the image plane of the camera.

Whereas, the previously known configuration (computed for previous camera frame or time step) of the articulated object, denoted with $\boldsymbol{q}$, affects the motion of *virtual visual features*, gathered in the vector $\boldsymbol{s(q)} \in \mathbb{R}^{\boldsymbol{2p}}$. Note that the virtual visual features depend on (i) the geometric model of the object, assumed to be known and used to calculate the location of the virtual PoI in $\boldsymbol{\mathcal{F}_c}$, and (ii) the camera intrinsic parameters, used to project the virtual PoI on the image plane. Since $\boldsymbol{\mathcal{F}_c}$ is the reference frame, we do not need the camera extrinsic parameters

to build the projection model.

Assuming known the object's PoI $\boldsymbol{\rho}$, and measuring the visual features $\boldsymbol{s}^*$, the objective of the tracking algorithm is to estimate the vector $\boldsymbol{q}^*$.

Using the previously known estimation $\boldsymbol{q}$, the model of the object is updated, so that the positions of the PoI $\boldsymbol{\rho}$ are also estimated and available for the subsequent computations. Then, for each visual point feature $\boldsymbol{i = 1, \ldots, p}$ the following actions are performed:

- the Jacobian $\boldsymbol{J_i}$ of the corresponding PoI is computed using $\boldsymbol{q}$ and $\boldsymbol{\rho_i}$. The exact Jacobian computation is explained later in this section;
- the virtual visual feature $\boldsymbol{s_i(q)}$ is obtained projecting 3D points $\boldsymbol{\rho_i}$ on the image plane, using the projection model of the camera;
- the interaction matrix $\boldsymbol{L_i}$ is computed using the estimated depth of the point (i.e., the $\boldsymbol{z}$-coordinate of the point $\boldsymbol{\rho_i}$) and the image plane coordinates of the point feature (recall Section 2.3);
- finally, the articulation matrix $\boldsymbol{A_i}$ is obtained.

Once all these operations are repeated for all the features, the articulation matrix is fully composed. Finally, $\dot{\boldsymbol{q}}$ is computed and $\boldsymbol{q}$ obtained by numerical integration, as described in the previous section (explicit Euler).

Let us explicitly describe the computation of the PoI's Jacobian, to be used in (3.5). To this end, we first need to express the Jacobian of the object's FB.

The orientation of the objet's FB in (3.1) is expressed with the unit quaternion $\boldsymbol{\sigma_o} = (\eta, \varepsilon)^T$, where $\varepsilon = (\varepsilon_1, \varepsilon_2, \varepsilon_3)$. From $\boldsymbol{\sigma_o}$, the rotation matrix $\boldsymbol{R_o}$ expressing the rotation from $\boldsymbol{\mathcal{F}_c}$ to $\boldsymbol{\mathcal{F}_o}$ can be reconstructed. Furthermore, from the rule of the quaternion propagation [61], it is possible to relate the angular velocity of the frame $\boldsymbol{\mathcal{F}_o}$ in its own frame to the derivative of the quaternion parameters:

$$^o\boldsymbol{\omega_o} = \boldsymbol{2E^T\dot{\sigma}_o} \tag{3.9}$$

$$\boldsymbol{E} = \begin{pmatrix} -\varepsilon^T \\ \eta \boldsymbol{I_3} - \boldsymbol{S}(\varepsilon) \end{pmatrix} \tag{3.10}$$

where $\boldsymbol{I_3}$ indicates the $\boldsymbol{3 \times 3}$ identity matrix and $\boldsymbol{S(\varepsilon)}$ is the skew symmetric matrix associated to ε. From (3.9), it follows that the velocity of the FB in the camera frame is

$$\begin{pmatrix} \boldsymbol{v_o} \\ \boldsymbol{\omega_o} \end{pmatrix} = \begin{pmatrix} \boldsymbol{R_o} & \boldsymbol{S(p_o)R_o} \\ \boldsymbol{O_3} & \boldsymbol{R_o} \end{pmatrix} \begin{pmatrix} \boldsymbol{I_3} & \boldsymbol{O_{3\times 4}} \\ \boldsymbol{O_3} & \boldsymbol{2E^T} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{p}}_o \\ \dot{\boldsymbol{\sigma}}_o \end{pmatrix} \tag{3.11}$$

that actually defines the Jacobian of the object's floating base:

$$\boldsymbol{J_o} = \begin{pmatrix} \boldsymbol{R_o} & \boldsymbol{S(p_o)R_o} \\ \boldsymbol{O_3} & \boldsymbol{R_o} \end{pmatrix} \begin{pmatrix} \boldsymbol{I_3} & \boldsymbol{O_{3\times 4}} \\ \boldsymbol{O_3} & \boldsymbol{2E^T} \end{pmatrix} \tag{3.12}$$

where $\boldsymbol{O_3}$ is the $\mathbf{3 \times 3}$ zeros matrix.

If the $\boldsymbol{i}^{\text{th}}$ PoI belongs to the FB of the object, then the corresponding Jacobian to be considered is

$$\boldsymbol{J_i} = \begin{pmatrix} \boldsymbol{I_3} & -\boldsymbol{S}({}^{o}\boldsymbol{\rho_i}) \\ \boldsymbol{O_3} & \boldsymbol{I_3} \end{pmatrix} \begin{pmatrix} \boldsymbol{J_o} & \boldsymbol{O_{6\times n}} \end{pmatrix} \tag{3.13}$$

where ${}^{o}\boldsymbol{\rho_i}$ is the coordinate vector of the $\boldsymbol{i}^{\text{th}}$ PoI expressed in $\boldsymbol{\mathcal{F}_o}$. On the other hand, if the $\boldsymbol{i}^{\text{th}}$ PoI is attached to the $\boldsymbol{l}^{\text{th}}$ link of the articulated object, than the corresponding Jacobian is

$$\boldsymbol{J_i} = \begin{pmatrix} \boldsymbol{I_3} & -\boldsymbol{S}({}^{l}\boldsymbol{\rho_i}) \\ \boldsymbol{O_3} & \boldsymbol{I_3} \end{pmatrix} \begin{pmatrix} \boldsymbol{J_o} & \boldsymbol{J_l} & \boldsymbol{O_{6\times(n-l)}} \end{pmatrix} \tag{3.14}$$

where ${}^{l}\boldsymbol{\rho_i}$ is the coordinate vector of the $\boldsymbol{i}^{\text{th}}$ PoI expressed in $\boldsymbol{l}^{\text{th}}$ link frame, and $\boldsymbol{J_l}$ is the Jacobian of the $\boldsymbol{l}^{\text{th}}$ link of the object ($\boldsymbol{l = 1, ..., n}$).

Experiments using PoI in the drawer opening use-case are presented in Chapter 4, Section 4.4.3.

3.5 Line feature based parameter estimation

With respect to the methodology, described in the previous sections, the following aspects need to be considered to enable line based parameter estimation in the proposed framework:

- Line representation on the image plane is given by a 2D vector of line polar coordinates, $\boldsymbol{(\theta, \rho)}$.
- The interaction matrix $\boldsymbol{L_i}$ in Equation 3.5 now represents the relation between the camera velocity and the velocity vector of the line polar coordinates $\boldsymbol{(\dot{\theta}, \dot{\rho})}$. Such a mapping is well studied and presented for example in [62];
- The visual feature Jacobian matrix form $\boldsymbol{J_i}$ defined in Equations (3.14)(3.13) must be adapted to the case of line features;
- While point feature 2D representation vector contains coordinates of the same type and scale, the line representation in polar coordinates contains two elements that are completely different in nature and scale ($\boldsymbol{\theta}$ is measured in degrees or radians, while $\boldsymbol{\rho}$ is measured in pixels). To address this issue, we can apply different gains ($\boldsymbol{\lambda_\theta}$ and $\boldsymbol{\lambda_\rho}$) for each entry of the error vector in Equation 3.7 as follows:

$$\dot{\boldsymbol{q}} = \underline{\mathrm{A}}^{+} \begin{pmatrix} \boldsymbol{\lambda_\theta} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda_\rho} \end{pmatrix} \boldsymbol{e} + \underline{\mathrm{A}}^{+} \dot{\underline{\mathrm{s}}}^{*} \tag{3.15}$$

3.6 Markerless feature extraction

To enable markerless edge based tracking, we need to match the edges extracted from the image to their corresponding counterparts in the 3D model (Figure 3.2). In this part of the work we describe how to achieve this goal. We present the main modules of the proposed framework, their interconnection and the structure of the overall tracker and the pose estimation system.

(a) Sample input. (b) Sample output.

Figure 3.2: Given an input image (a), the output of the markerless feature extraction algorithm is a coarse detection of the bounding box around the object (green box in (b)), and the edges correctly assigned to the corresponding classes on the model (different color-labeled lines in (b)).

The general scheme of the system is presented in Figure 3.3. The *Learning* algorithm is adapted to learn to discriminate between known object features (e.g. lines that form object model). It takes as an input the object's model afforded with the best geometric features and other parameters to best tune both the detection and the tracking. The output of the learning process is a detection and prediction models to detect coarse position of the object in the *current image* and predict possible positions of the known landmarks. The *detector and predictor* can act on *initialization request* and output the *table of object features* with their location on the *current image* and initial values of the table data structure. This feature table is the core of the entire process as it is also used by the *guided Hough* line fitting module and is continuously updated by the *edge selection and update* blocks. The *guided Hough* detects exact lines in the *current image*, while being guided by the estimation provided by both the *detector and predictor*, and in the same time –could be at different update rates, the *feature table* update of the previous iteration. The latter is a prediction provided from the results of the *tracking and pose estimation* module. Moreover, the features in the *feature table* are evaluated and the minimum number of the features required to estimate the pose is selected by the *edge selection* module; the feature table is updated accordingly. The output of the *guided Hough* module is a set of edge parameters, which is passed as an input to the *tracking and pose estimation* module. The pose of the object (here the printer) is updated based on the visual information and is passed as a feedback

to the MQP control. The joint update for the "multi-robot" system is computed, executed by the robot (here the HRP-4). The robot then captures the next camera frame and the entire process is repeated until the tasks objectives are achieved.

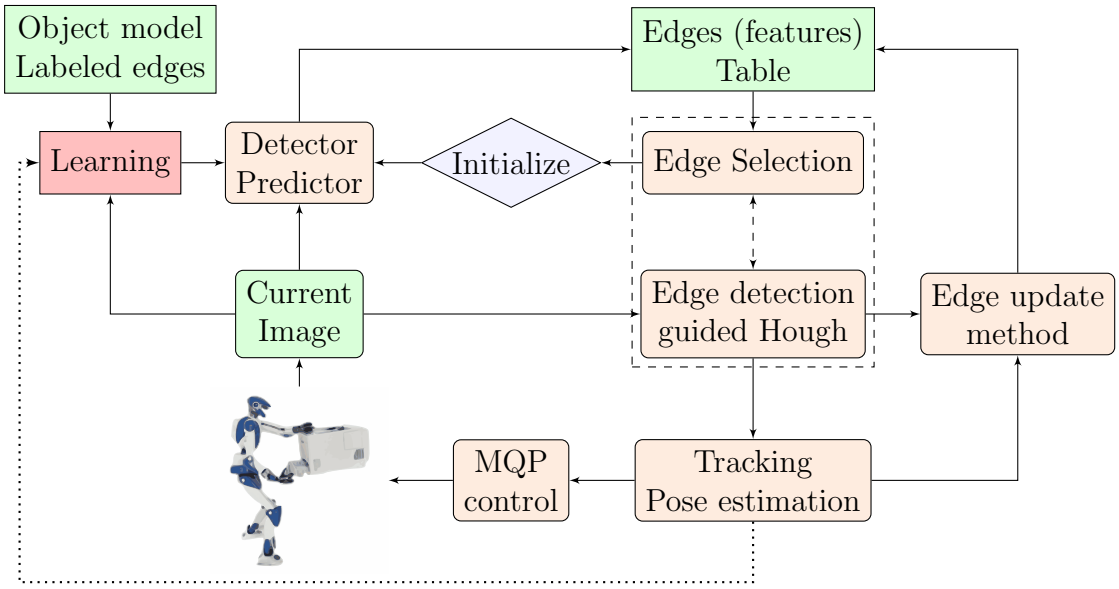


Figure 3.3: The general scheme of all modules.

The proposed framework also has a potential of being augmented with the continuous learning from experience functionality, when the result of successful tracking is automatically annotated for the supervised learning process and used to enhance detector and predictor models. In the following sections, all the modules (blocks), the data flow exchanged and the data structure are described in more details together with their algorithms and eventually their implementation.

3.7 Detector and Predictor modules

We need to derive approximate position of the object model edges on the image at the (re)initialization stage, i.e. when the tracker information from the previous iteration is either completely missing or is not reliable. We use several machine learning techniques, that can provide robust, reliable and real-time initialization solution. The key to achieving real-time performance of initialization step is to use fast coarse bounding box type object detector prior to any further processing as it greatly reduces the amount of data that needs to be processed.

Historically, first real-time performance for the object detection has been achieved by Viola and Jones in 2001 [2]. Since 2005, histogram-of-orienter gradient (HOG) proved to be a fast and reliable feature descriptor for various object detection tasks [5][63][64]. The HOG based object detectors take less time to train than

Viola-Jones, they provide real-time performance and good detection accuracy. It is important to note, that accuracy of the detection will drop if the object appearance varies significantly in the training set and/or during the test detection. The convolutional neural networks (CNN) can deal with such issues better, but it rarely achieves real-time performance, while maintaining high detection accuracy, especially if executed on CPU. Thus, HOG based method was selected for coarse detection. The idea of this method is that a feature descriptor for an object class is created based on training samples. The HOG operation computes dominating local gradient directions to produce a descriptor for object detection. A sample descriptor of the floating base of a printer is shown in Figure 3.4. The pseudo-algorithm for computing HOG descriptor is presented in Algorithm 2.

(a) HOG descriptor. (b) Training sample.

Figure 3.4: (a) Learned HOG filter for a printer floating base, structure of the printer is recognizable, (b) Sample from detector training set.

Algorithm 2: HOG descriptor computation

Data: `CurrentImage`
Result: `HOGdescriptor`

$\boldsymbol{g_x}, \boldsymbol{g_y} \leftarrow$ ComputeHorizontalVerticalGradients(`CurrentImage`)
gradMag, gradDir $\leftarrow$ allocateSpace()
foreach `pixel in CurrentImage` **do**
  gradientMagnitude[pixel] = $\sqrt{\boldsymbol{g_x^2} + \boldsymbol{g_y^2}}$ ▷ compute gradient magnitude
  gradientDirection[pixel] = $\textbf{arctan}\,\dfrac{\boldsymbol{g_y}}{\boldsymbol{g_x}}$ ▷ compute gradient direction
end
HOGdescriptor $\leftarrow$ allocateSpace()
imageBlocks $\leftarrow$ DivideIntoBlocks(`CurrentImage`)
foreach `b in imageBlocks` **do**
  ▷ compute histogram for 9 discrete classes of gradient orientation
  HOGdescriptor[b.id] $\leftarrow$ getHistogram(`b, gradMag[b.id], gradDir[b.id]`)
end

The object detector is trained given a set of images and corresponding manually annotated positions of bounding boxes, $(\boldsymbol{I_i}, \boldsymbol{B_i})$. For each sample in the training set its HOG descriptor is computed according to Algorithm 2. Areas of the training samples outside the annotated bounding box are used to generate negative samples and compute corresponding HOG descriptors. Linear support vector machine (SVM) is trained on those sample HOGs. After the model is trained, given a new image, $\boldsymbol{I}_{\text{test}}$, sub-images of various positions and scales are evaluated by trained SVM to make a decision whether given sub-image contains object of interest or not.

After coarse detection is performed successfully and we identified smaller part of the image, which contains an object of interest, more fine detection can take place. To better estimate the exact shape of the object as it appears in the image, few known auxiliary object landmark positions inside the bounding box are predicted. This is done using combination of appearance and shape knowledge learned from sample data. According to the method, proposed in [65], given a training set of images, bounding boxes and landmark positions, $(\boldsymbol{I_i}, \boldsymbol{B_i}, \boldsymbol{S_i})$, a cascade consisting of $\boldsymbol{T}$ strong regression functions, $\{\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{T-1}}\}$ is learned. Each strong regression function in the cascade is trained to improve upon prediction of the previous function in the cascade. Each strong regressor is a linear combination of $\boldsymbol{K}$ weak regression tree models, which are trained using gradient boosting method to predict update vector, which needs to be added to the current guess of the landmark positions in order to achieve better alignment with the actual object shape, which is observed in the image.

The mean position of all landmarks is computed over the entire training set.

$$\boldsymbol{S}_{\text{mean}} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i}} \boldsymbol{S_i^{B_i}}, \tag{3.16}$$

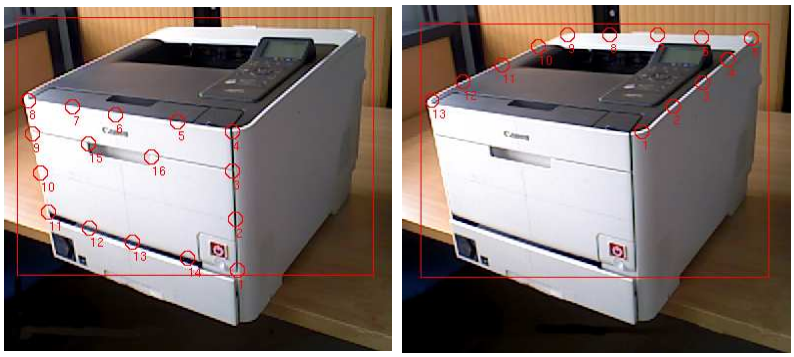
where $\boldsymbol{S_i^{B_i}}$ is expressed in the corresponding bounding box coordinate system and not in the full image coordinate system. The $\boldsymbol{S}_{\text{mean}}$ serves as an initial guess of the landmark positions in the bounding box, $\boldsymbol{B}_{\text{test}}$, detected in the test image $\boldsymbol{I}_{\text{test}}$. In order to correct this simple initial guess, iterative process of evaluating the cascade of learned regression functions $\{\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{T-1}}\}$ and applying resulting update vectors is preformed as follows:

$$\begin{aligned}
&\boldsymbol{S_1} = \boldsymbol{S}_{\text{mean}} + \boldsymbol{r_0}(\boldsymbol{B}_{\text{test}}, \boldsymbol{S}_{\text{mean}}) \\
&\boldsymbol{S_2} = \boldsymbol{S_1} + \boldsymbol{r_1}(\boldsymbol{B}_{\text{test}}, \boldsymbol{S_1}) \\
&\ldots \\
&\boldsymbol{S}_{\text{predicted}} = \boldsymbol{S_{T-1}} + \boldsymbol{r_{T-1}}(\boldsymbol{B}_{\text{test}}, \boldsymbol{S_{T-1}})
\end{aligned}$$

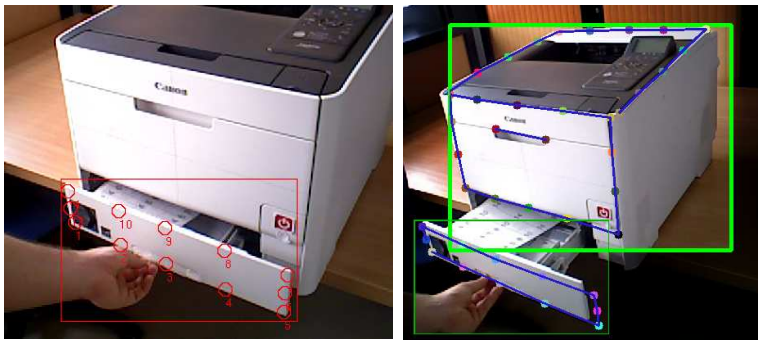
After final $\boldsymbol{S}_{\text{predicted}}$ is calculated, it is regarded as a prediction of landmark positions in the current frame. We use this prediction to validate existence of the actual image feature around the predicted image area. If result of the validation

process is positive, we extract the precise parameters of actual image feature, that can be further used to compute vision based control input for a robot to manipulate the observed object. Details of this process are presented in the following sections.

To test performance of object detector and landmark position predictor for the articulated object used in our manipulation experiment, 140 images from a sample recording were annotated. The bounding box around floating base of the object was indicated on every image. The floating base was used rather than the entire object to reduce variance in appearance of the training samples. For the same reason, the detector for the drawer was trained separately. The front and upper faces of the printer were selected for landmark annotation, due to their more textured appearance 16 and 13 landmarks were annotated on those faces respectively. Additional 12 landmarks were annotated on the drawer. Annotation example is presented in Figures 3.5a 3.5b 3.5c. A sample result of the detection and landmark position prediction is shown on Figure 3.5d.



(a) Front face landmarks. (b) Upper face landmarks.



(c) Drawer landmarks. (d) Result.

Figure 3.5: (a), (b), (c) annotated landmarks, and (d) ample detection/prediction result frame.

Dlib toolkit[2] implementation of the object detector framework [5] was used to train SVM for coarse object detector and implementation of [65] was used to train a cascade of regressors for landmark position prediction. The SVM regularization

[2]https://github.com/davisking/dlib

parameter, $\boldsymbol{C}$, was set to **15**, to enable better separation of the positive and negative samples. For the landmark position prediction model training, due to the small amount of annotated data, parameters of the model were changed to enable better generalization. The number of initializations used per image in the training set to generate data for regressor function training was set to **300**. A regularization parameter used in gradient boosting, called learning rate or shrinkage factor, denoted by $\boldsymbol{\nu}$, was set to **0.05**, as it has been shown empirically in several works on gradient boosting, that lower values of $\boldsymbol{\nu}$ greatly decrease test set error [66] and thus allow model to generalize better. The depth of each individual regression tree model for landmark position prediction was limited to **2**. A combination of those parameter values has shown to yield subjectively best alignment on the image plane with the 3D object model on two sample videos, which in total contain **7000** frames, which were not used for the training of the detector or predictor models.

The output of the initialization process must be in a form of approximated edges parameters. From the sample result of detector and predictor, shown in Figure 3.5d, we see that all necessary components for computing reliable initial values for the sub-set of object edges are available.

First, landmarks are gathered into known groups, which lie on the same edge. Then, for each group number of visible landmarks is computed, if this number is less than 2, this edge is considered to be not visible. If number of visible landmarks per edge is two or more, those points are used to compute edge parameters required by the tracker. The algorithm and details of this process are discussed in Section 3.8.3. Brief visualization of the process is shown on Figure 3.6.

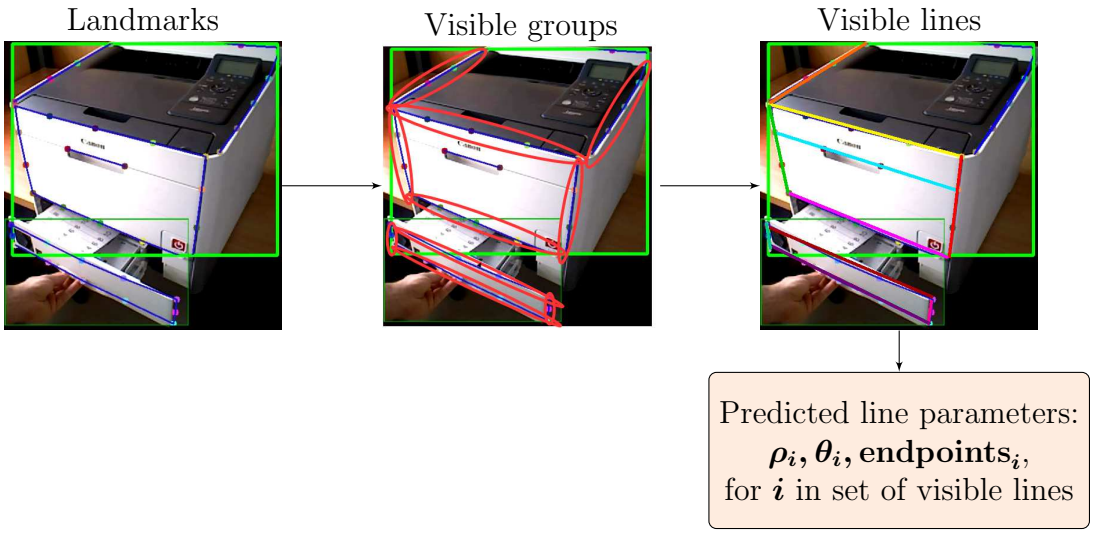


Figure 3.6: The scheme of the initialization process from the output of detector/predictor modules.

It is important to note that the output of initialization method is not used in the control. Initialization method only provides prediction of where possible lines may be in the frame, this prediction is used further by precise image processing guided Hough method to verify existence of actual line near predicted location and to compute the parameters of actual line, which is then passed to the tracker module to compute input for the MQP control.

3.8 Edge selection and update method

The edge selection and update method is at the heart of the overall pose estimation problem. It bridges the detector and predictor to the tracker through several intermediate modules with various internal closed-loops as illustrated in Figure 3.3.

The edge selection and the edges update share a common data structure. The detector and predictor modules create the data structure and fill it with initial values. The edge selection picks parameters for the guided-Hough to seek for a minimal set of quality edges to be given to the tracker. The latter estimates the pose and provides a prediction on the tracked edges that are used to update the table. Hence, the update of the Edges (features) Table is made from the tracker and also from the detector and predictor. In the following, we describe what are the fields that compose the Edges (features) Table.

3.8.1 The edges (features) table

At each initialisation request, the *detector and predictor* modules create the table using the *current image* and the data from the 3D poly-articulated object model as an input. If the table exists already, the reinitialisation consists in filling the table only from the current image, since the data concerning the 3D poly-articulated object model is already in the table.

The structure of the Edges Table is composed by three categories of data: (i) those that can be filled directly from the 3D model of the objects, (ii) those using the latter structure in (i) and completed from the current image, and finally (iii) those that are computed from previous data (i) and (ii).

The following fields are the structure data obtained from the 3D model of the object and affordance knowledge (we specify only one level of parent dependancies in the data structure for the clarity of the reading and writing):

- `edgeID`: (unsigned integer, char or string) it is the edge label. The edges are potentially used for tracking and they come from the 3D CAD model. That is to say, the model of the objects already says what are the edges that are worth to be tracked. The 3D CAD model can be rich in edges

but not all of them (namely the internal hidden ones, or those representing insignificant details) are worth to be labelled to be tracked. So there is an off-line effort where the user (or an algorithm in the future work) elects the edges (line segments) from the mesh composing the 3D CAD model of the poly-articulated object.

- `linkID`: (unsigned integer, char or string) we assign to each poly-articulated object's link an ID (one can take that of the `urdf` file if any, otherwise we can store such an ID in the description of the object). To each `linkID` we assign a set of edges identifiers.
- `linkID.nbrPreferenceTracking`: (unsigned int) stating for each `linkID` the total number of `preferenceTracking`.
- `linkID.preferenceTracking`: (an array of lists of ordered edge(s) for each link), list of the preference for tracking (for example, for given link, we can order the preferred edges for the tracking, or their combination if more than one is needed for a given link). The best scored `preferenceTracking` are selected for the tracking). The ordering of this array is made when other fields are filled.
- `preferenceTracking.nbrEdges`: (unsigned int) stating for each `preferenceTracking`'s list, the number of edges present in it.

The following fields are obtained from the current image, after the previously mentioned fields are present in the table. These fields are related to the edges and their inner data structure:

- `edgeID.rho` and `edgeID.theta`: (floats) are the coefficient of the normal representation of the lines (that will be used by the guided Hough module). These data are roughly estimated from the detector and predictor at the creation of the table. Then, for those edges that are tracked, the update of these values comes from the prediction as an output of the tracker module; for those who are not, the update come from the detector and predictor. Ideally, any update shall be made by a combination of the detector/predictor and the tracker.
- `edgeID.points[2]`: (array of two elements of 2D points), these points can be used for plotting purposes and also to compute the length of the edge as it appears in the screen.
- `edgeID.isVisible`: (boolean) if the edge is visible or not in the image; this is obtained directly from the detector/predictor at the initialization and assessed continuously by the tracking process when true.
- `edgeID.isTracked`: (boolean) says weather a given preference edge is tracked or not. At the initialization this field is set to `false` for all the preference edges for a given `preferenceTracking`.

The following fields are the data structure derived from the previous data and concern marking and other data that are useful for the edge selection for tracking:

- `edgeID.edgeLength`: (float) each edge has a length obtained from `Points[2]` (i.e. the 2-norm).
- `edgeID.dX` and `edgeID.dY`: (float), we assign to each edge a distance metric to the image borders: it is the distance of a 2D edge to the border of the image (if any of this distance is zero, then the edge is touching the image border).
- `edgeID.qualityEdge`: (float) is a mark given to an edge to be determined from a formula that will be discussed later.
- `preferenceTracking.qualityTracking`:(float) is a mark given to `preferenceTracking` from `qualityEdge` to be discussed later in this section.

These are the fields that constitute the Edge Table. In the following we will first discuss how the edges are selected assuming the Table already fully filled with right values.

3.8.2 Selection of edges for tracking

The edge selection is made easy from the Edges Table data. Indeed, we have already put efforts and emphasized on the importance of having a well documented Edge Table in order to make available the data for the tracking. The edge selection algorithm will seek for the `preferenceTracking`, i.e. potentially a set of elected edges, that are tracked ans have the highest mark. In the table, for each `linkID` its `preferenceTracking` sets are ordered in a first in, first out perspective, from the tracked ones to the lower marked ones. For each link, we always provide the first `preferenceTracking` list to the *Hough-guided edge detection* module. If the latter succeeds in returning the edge(s), they are sent to the tracker. Otherwise, the next `preferenceTracking` in the list is selected and evaluated, etc. If all `preferenceTracking` for a given link were not detected by the *Hough-guided edge detection*, then there are two possible options:

1. the link without any Hough-guided edges found is not visible (but this is in opposition with the detector/predictor that founds them);
2. the detector/predictor returns bad information (maybe the image changed a lot already), here we need to infer initialization of the Edges Table.

If the tracker cannot be started after a certain number of initializations, the pose estimation fails severely and one needs to reconsider the entire approach by refining the parameters for the object in question.

In the other hand, if edges are found by the Hough-guided process, their parameters are sent to the *Tracking and pose estimation* module and in the same time to the *Edge table update* modules.

The entire pseudo-algorithm of Edges selection is presented in Algorithm 3.

Algorithm 3: Edge selection

```
Data: CurrentImage, EdgeTable
Result: edgesList, the list of edges to send the tracker
scoring(edgeID) ▷ Call Algorithm 4, we score at this level, right before usage
foreach linkID do ▷ we screen all links
    ▷ find the requested number of preferenceTracking for this link
    for (i = 1 : nbrPreferenceTracking) do
        test ← GuidedHough(CurrentImage, preferenceTracking[i], edge) ▷ run guided hough for the preferenceTracking[i] if all found, return true otherwise return false
        if test then ▷ add the edge parameters to the list
            edgesList.add(edge)
            exit ▷ found all the edges in preferenceTracking[i]
        else
            i ← i + 1 ▷ Next preferenceTracking in this link's list
        end
    end
    if (i > linkID.nbrPreferenceTracking) then ▷ no edge found for this link]
        Initialization(CurrentImage, EdgeTable)
        nbrInitialization ← nbrInitialization +1
        if max(nbrInitialization) reached then
            user warning for/and robot safe mission cancelling
        end
    end
end
```

In Algorithm 3, the function "GuidedHough" will take as input each of the current i^{th} `linkID.preferenceTracking` edges parameters to seek for that edge Hough parameters in the image portion where the edge is expected to be. This function is detailed in Section 3.9. The `edgesList.add` method only copy the result of the GuidedHough function returned in variable "edge" into the edges list to be passed to the tracker. The "Initialization" function is explained in Section 3.8.3.

Algorithm 4 is used to score the edges and the preference tracking so that they are ordered by merit (best first) as to ease the Edge selection later.

Algorithm 4: Scoring

Data: `Edge Table`
Result: `edgeID.quality` and `preferenceTracking.qualityTracking` updates

foreach `edgeID` **do** ▷ start by marking edges
  `dX` ← min(`point[1].X`, `point[2].X`, imageSize(X) - `point[1].X`, imageSize(X) - `point[2].X`)
  `dY` ← min(`point[1].Y`, `point[2].Y`, imageSize(Y) - `point[1].Y`, imageSize(Y) - `point[2].Y`)
  `edgeLength` ← norm(`point`)
  `qualityEdge` ← scoreEdge(`edge.ID`)
end
forall the `linkID.preferenceTracking` **do** ▷ mark each `preferenceTracking` in each `linkID` from the marks of its edges
  `qualityTracking` ← scorePreferenceTracking(`preferenceTracking`)
end
foreach `linkID` **do** ▷ sort `preferenceTracking` by merit
  order(`preferenceTracking[]`)
end

Now we propose a simplistic score strategy for the edges described in Algorithm 5. The idea is to favor edges that have good visibility through the parameter `edgeID.length` modulated by a user-given gain $\boldsymbol{\alpha}$, and those who are far from the image borders. For the latter, the score is penalized thanks to `edgeID.dX` (modulated by $\boldsymbol{\gamma}$ that can be taken as a gain divided by the size of the image along its X axis) and `edgeID.dY` (modulated by $\boldsymbol{\lambda}$ that can be taken as a gain divided by the size of the image along its Y axis). More sophisticated scoring can be envisioned, for example, we can consider the noise of the edge, or consider the past score to increase the current one, add threshold to previous parameters, etc. But since we do closed-loop control, we better consider simple scoring strategies and computations.

Algorithm 5: function scoreEdge(`edgeID`) in Algorithm 4.

Data: `edgeID`
Result: `edgeID.quality` updates

if `isVisible` **then**
  quality ← $\boldsymbol{\alpha}$ `edgeLenght` + $\boldsymbol{\gamma}$ `dX` + $\boldsymbol{\lambda}$ `dY`
else
  quality ← $-\mathbf{1}$
end

Now that we have scored the edges we will devise a simple formula to score the `preferenceTracking`. This is explained in Algorithm 6. If in a

given `preferenceTracking`, one of the `edgeID.quality` $< \mathbf{0}$, then the entire `preferenceTracking` is considered bad, otherwise, we can simply assign `preferenceTracking.quality` to be $\sum_{\texttt{.nbrEdges}}$ `.edgeID.quality` i.e. the numerical sum of all edge scores composing a given `preferenceTracking`. We can of course modulate this by dividing the sum on the number of edges or not; we can also consider other aspects.

Algorithm 6: function scorePreferenceTracking(`preferenceTracking`) in Algorithm 4.

Data: `preferenceTracking`
Result: `preferenceTracking.qualityTracking` updates

```
qualityTracking ← 0 ▷ Initalization
for i = 1 : nbrEdges do
    if edgeID.quality > 0 then
        qualityTracking ← qualityTracking + edgeID.quality
    else
        qualityTracking ← −1
        return
    end
end
qualityTracking ← qualityTracking/nbrEdges ▷ eventually take a mean
```

3.8.3 Edge table initialization and updates

The update of the Edge Table is made in two parts (eventually not at the same frequency) by the Detector Predictor module and the Edge Update Method (outcome of the Edge Selection guided Hough results and the tracker).

Prior to the update, we have the initialization phase, that can also be requested when the tracker or the edge selection algorithm is stacked. Initialization of the Edge Table by the detector predictor module is done according to Algorithm 7. Here we describe this algorithm. If the Edge Table is empty, it is first filled with initial values. For every link of the object bounding box around the object in the current image is detected. The cropped bounding box, which contains the poly-articulated object (entirely or partially), is passed to the landmark predictor to estimate possible position of the link landmarks. Known landmark groups form edges on the link, those groups are known for each link from object description. For each such group of landmarks, visibility of a possible link edge is evaluated. If at least two landmarks from the group are visible, then they can be used to derive edge data, which is necessary to fill the edges data. If landmarks, which are supposed to form the edge are not visible, then this edge is set to be non-visible

in the Edge Table.

Algorithm 7: Initialization

```
Data: CurrentImage, EdgeTable, ObjectDescription
Result: EdgeTable updated

if EdgeTable is empty then ▷ first step: build initial table from the object
model; done once
    EdgeTable ← BuildEdgeTable(ObjectDescription)
end

for (linkID = 1 : ObjectDescription.numberOfLinks) do ▷ second step:
update edges with true value as available from the current image]
    objectWindow ← DetectObject(CurrentImage, linkID)
    landmarks ← PredictLandmarks(objectWindow, linkID)
    landmarkGroups ← getLandmarkGroups(linkID,
    ObjectDescription)
    foreach group in landmarkGroups do ▷ fill the edges data
        edgeID ← getEdgeID(linkID, group)
        if groupIsVisible(group,landmarks) then
            edgeID.theta, .rho, .points ← getEdgeData(group,
            landmarks)
            edgeID.isVisible ← true
        else
            edgeID.isVisible ← false
        end
    end
end
```

In case of successful tracking, each edge of the object model can be updated in the Edge Table. Since tracking computes full pose of the object in 3D, projection of this information onto the image can yield very rich information even about fully-occluded edges. The information for each edge, including it's `theta`, `rho`, and `points` parameters, can de derived from tracker results along with additional information for `isVisible`, `isTracked` field of the Edge Table.

Another important issue of the tracking process is the update. The update occurs at two levels: (i) the update that outcome from the tracker, and (ii) the update that outcome from the detector predictor.

The first is a continuous update of the edge parameters directly from the tracker. Indeed, since these edges are tracked, their update is of higher priority and of better reliability. Therefore, the *Edge update* method will update all the edges that are tracked with predictive values obtained from the tracker (input: `edgeList`, and output: edges' parameters derivatives), in a closed-loop form. We emphasize that the update of the tracked edge is not the parameters values obtained from the

current image, but the predicted ones for the next iteration. This is described thoroughly in Algorithm 8.

Algorithm 8: Edge Update from Tracker and Edge Selection

Data: `edgesList` from Algorithm 3, and $(\dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\rho}})$ from the tracker
Result: `edgeID` updates in `edgesList`

1 `edgesList`.$\begin{bmatrix} \dot{\boldsymbol{\rho}} \\ \dot{\boldsymbol{\theta}} \end{bmatrix} = -\boldsymbol{L}\boldsymbol{J}\dot{\boldsymbol{q}}$ ▷ derivatives of the tracked edges' parameters
forall the `edgeID` $\in$ `edgesList` **do**
2 `edgeID.rho` $\leftarrow \boldsymbol{\delta}$ `edgesList`[`edgeID`].$\dot{\boldsymbol{\rho}}$ + `edgeID.rho` ▷ predicted $\boldsymbol{\rho}$
3 `edgeID.theta` $\leftarrow \boldsymbol{\delta}$ `edgesList`[`edgeID`].$\dot{\boldsymbol{\theta}}$ + `edgeID.theta` ▷ predicted $\boldsymbol{\theta}$
4 `edgeID.point` $\leftarrow$ 2dLineProject(`edgeID.point`, `edgeID.rho`, `edgeID.theta`) ▷ we obtain the predicted points by projecting the current ones on the 2D line obtained from the predicted parameters (the previous two instructions).
end
5 copy `edgesList` data to Edge Table

In Algorithm 8, $\dot{\boldsymbol{\rho}}$ and $\dot{\boldsymbol{\theta}}$ are the derivative of the tracked edges that can be computed directly from the tracker data as explained in line 1, in which: $\boldsymbol{L}$ is the interaction matrix, $\boldsymbol{J}$ is the Jacobian and $\dot{\boldsymbol{q}}$ is the vector of the generalized poly-articulated object joints, $\boldsymbol{\delta}$ is a constant value taken as the sampling time $\boldsymbol{\Delta t}$. Since we have the derivative of the edge parameters we compute their prediction for the next iteration. This is done in lines 3 and 4 respectively. From here we can compute using the line formula $\boldsymbol{\rho} = \boldsymbol{y}\cos(\boldsymbol{\theta}) + \boldsymbol{x}\sin(\boldsymbol{\theta})$ the predicted line and we can then project the current edge's points onto that new line to have the prediction of the next iteration 2D point coordinates, line 5. When all is done, the edgesList contains predictive information of the tracked edges that can be used to update the Edge Table, line 7.

The second level of update deals with all the edges that are not tracked but for which we need to keep track of the changes in the current image. This update is the outcome of the *detector and predictor* module without intialization. That is to say, in a separate thread, the *detector and predictor* updates all `edgeID` that are not tracked. The update from the detector predictor is given by Algorithm 9 and

follows somehow the same steps as described in the initialization Algorithm 7.

Algorithm 9: Edge Update from Detector Predector

```
Data: CurrentImage, EdgeTable, ObjectDescription
Result: EdgeTable updated
forall the linkID do
    objectWindow ← DetectObject(CurrentImage, linkID)
    landmarks ← PredictLandmarks(objectWindow, linkID)
    landmarkGroups ← getLandmarkGroups(linkID,
    ObjectDescription)
    foreach group in landmarkGroups do
        edgeID ← getEdgeID(linkID, group)
        if edgeID.isTracked = false then ▷ only for non-tracked edges
            if groupIsVisible(group,landmarks) then
                edgeID.theta, .rho, .points ← getEdgeData(group,
                landmarks)
                edgeID.isVisible ← true
            else
                edgeID.isVisible ← false
            end
        end
    end
end
```

3.9 Hough-guided line detection

In classical Hough line fitting process, every possible value of $\boldsymbol{\rho}, \boldsymbol{\theta}$ parameters is evaluated for detecting a possible line in the image. Since we already have good approximation of line parameters in the Edge Table, either from initialization process or from the tracker update, we use this information to only evaluate small subset of possible line parameters. This is done by "guiding" Hough transform with approximated values inside sub-image, where edge candidate in predicted to be in the current frame. The Hough-guided system serves as an extractor of real image line parameters and as a validator that those predicted edges actually exist in the image.

The Hough-guided algorithm works in close interaction with(in) the edge selection Algorithm 3, described in Section 3.8.2. The Hough-guided detection takes as an input the `preferenceTracking` edges suggested by the edge selection process (Algorithm 3), and seeks for those edges in the current image, processed for edges extraction using Canny or other method, but only in a subscribed sub-image area. The latter is computed from a slightly enlarged area around the current edge points `preferenceTracking.edgeID.points`. The Hough-guided simply seeks for the

edge using Hough transform at the neighbourhood of the provided edge parameters `preferenceTracking.edgeID.rho` and `preferenceTracking.edgeID.theta`, see Algorithm 10. Thus it is being "guided" by approximation of those parameters as given in the Edge Table.

Algorithm 10: Hough-Guided algorithm

```
Data: CurrentImage, preferenceTracking
Result: returns true if all edges in preferenceTracking, false otherwise
foreach edgeID in preferenceTracking do
    if edgeID not already found then
        subImage ← crop(CurrentImage, edgeID.points)
        edge = Hough(subImage, edgeID.rho, edgeID.theta)
        if not empty(edge) then
            ▷ real edge found in the image near predicted parameters
            edgeID.isTracked ← true
            edgeID.isVisible ← true
        else
            edgeID.isTracked ← false
            edgeID.isVisible ← false
            return false
        end
    end
end
return true
```

3.10 Conclusion

In this chapter, we have presented the methodology for articulated pose tracking from primitive visual features, such as points and edges. We have proposed a framework, which combines advantages of a real-time performance of the machine learning based pose prediction and accuracy of the Hough-guided line extraction to enable markerless visual feedback for edge based articulated pose tracking.

The experimental results for the closed-loop object manipulation using presented methodology based on point features are described in detail in Section 4.4 of the following chapter.

Chapter 4

Integration, experiments and results

In the previous chapters of this work, we have introduced the theory behind vision based whole-body motion control in the QP framework (Chapter 2) and the poly-articulated object configuration estimation by the means of VVS based tracking (Chapter 3) for the use in a multi-robot QP control framework (MQP).

In this chapter we present the MQP and the integration of the described methods to perform real challenging experiments using the HRP-4 humanoid platform. We assess our developments and discuss the experimental results that have been obtained for each experiment together with the limitations and other aspects that might require additional efforts in the future work.

4.1 The multi-robot QP control framework: description and integration

The QP control framework, described in Section 2.2, is extended to the case of multi-robot QP (MQP). In this new version of the control framework, the fully or partially actuated robotic platforms, the environment and the rigid or non-actuated articulated objects are all modelled as a single *multi-robot system*, whose components interact by means of contact forces or mutually shared constraints.

Let us consider the case of the drawer opening operation. The multi-robot system in this case consists of the ground and the table modelled as a static environment, the HRP-4 humanoid robot with actuated joints, and the printer with one non-actuated prismatic joint for the drawer (also modelled as a robot with no actuator). The corresponding components are highlighted in Figure 4.1.

In this multi-robot system case, there are 3 components which interact with one

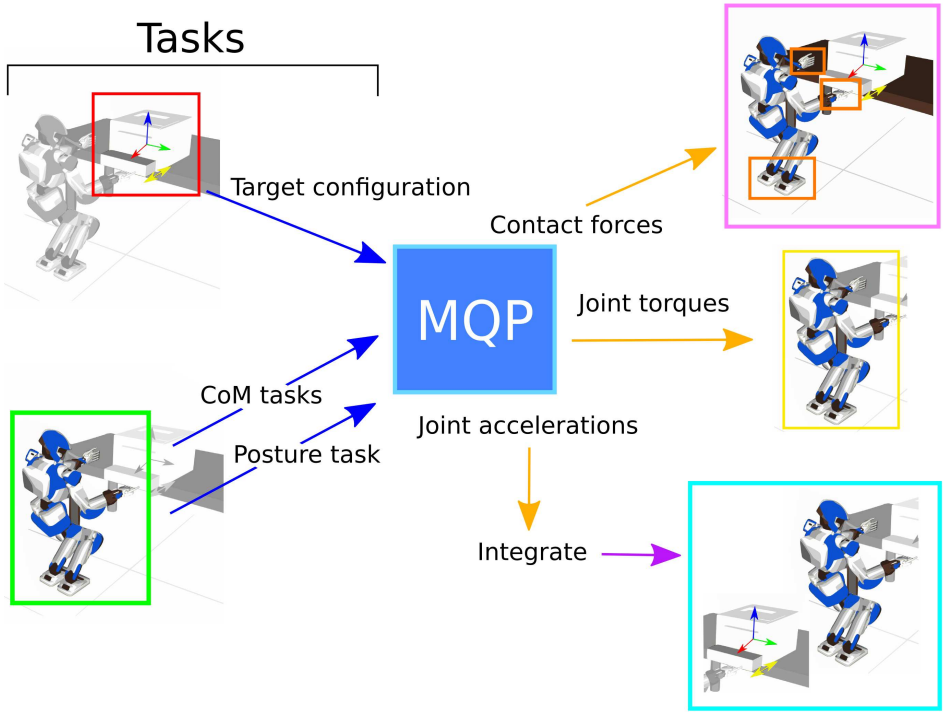


Figure 4.1: Given set of tasks, the MQP framework computes control solution for the overall state of the multi-robot system.

another through interaction contact forces: ground and table are considered static, the humanoid, and the articulated object (printer) with non-actuated prismatic drawer joint: they are considered as "robots". For the purpose of generality, suppose there are $\boldsymbol{n}$ number of "robot" components in the multi-robot system of the MQP. At every time step of the interaction, for all robot components in such a multi-robot system, their individual equations of motion ($\mathbf{EoM}_i$, $\boldsymbol{i = 1, \ldots, n}$, the same as Equation 2.10 in Section 2.2):

$$\boldsymbol{M_i(q_i)\ddot{q}_i + C_i(q_i, \dot{q}_i) + G_i(q_i) = S_i\tau_i + J_i^T f_i} \tag{4.1}$$

are combined (stacked) together through the interaction contact forces (or eventually other interaction constraints) to form a single EoM for the entire multi-robot system.

The desired configuration of the manipulated object is specified at the high level task space (e.g "robot opens the drawer by **10**cm"). The MQP framework finds optimal contact forces, joint torques and accelerations for the entire multi-robot system to achieve the desired configuration. To fulfil the tasks in a closed-loop control scheme, at every time step, the MQP needs to know the current state of the entire system for every component. Here, the MQP system is explained in the context of non-actuated articulated object manipulation. While the robot joints have encoders, which provide joint value at every control time-step, such transducers are absent on common objects we manipulate daily. Thus, it is necessary

to estimate the configuration of the manipulated object. In our work, computer vision is used for this purpose, as we already described in Chapter 3. From now on, "robot" means a real robot or articulated object, and we write it without quotes.

For the sake of completeness of MQP framework description, we give here the changes w.r.t single QP presented in Section 2.2. For more details, the reader may refer to [67][68]. The vector $\boldsymbol{f_i}$ stacks all contact forces applied on the surfaces of robot $\boldsymbol{i}$. They come in pairs of action/reaction forces among the system of robots according to Newton's third law, and opposite forces applied by the robot $\boldsymbol{i}$ on the robot $\boldsymbol{j}$ appear in the vector $\boldsymbol{f_j}$. Therefore $\boldsymbol{f_i}$ is decomposed into $\boldsymbol{f_i} = (\boldsymbol{f_i^0}, \boldsymbol{f_i^-}, -\boldsymbol{f_i^+})$ such that $\boldsymbol{f_i^0}$ are the forces applied by the static environment on the robot $\boldsymbol{i}$, $\boldsymbol{f_i^-}$ are the forces applied by the robots $\boldsymbol{j} < \boldsymbol{i}$ on the robot $\boldsymbol{i}$, and $\boldsymbol{f_i^+}$ are the forces applied by the robot $\boldsymbol{i}$ on the robots $\boldsymbol{j} > \boldsymbol{i}$. Let $\boldsymbol{F^0}$, $\boldsymbol{F^-}$, $\boldsymbol{F^+}$, be the vectors stacking all $\boldsymbol{f_i^0}$, $\boldsymbol{f_i^-}$, $\boldsymbol{f_i^+}$ respectively, and $\boldsymbol{K}$ the total number of forces in $\boldsymbol{F^-}$, i.e. such that $\boldsymbol{F^-} \in \mathbb{R}^{\boldsymbol{3K}}$. By virtue of Newton's third law, there exists a permutation matrix $\boldsymbol{\Pi} \in \mathbb{R}^{\boldsymbol{K}\times\boldsymbol{K}}$ such that

$$\boldsymbol{F^+} = (\boldsymbol{\Pi} \otimes \boldsymbol{I_3})\boldsymbol{F^-} \tag{4.2}$$

where $\otimes$ denotes the Kronecker product. Let $\boldsymbol{\Psi} = \boldsymbol{\Pi} \otimes \boldsymbol{I_3}$. The permutation matrix $\boldsymbol{\Psi}$ is decomposed into selection matrix blocks $\boldsymbol{\Psi_i} \in \mathbb{R}^{\boldsymbol{3K_i}\times\boldsymbol{3K}}$ in the form:

$$\boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\Psi_1} \\ \vdots \\ \boldsymbol{\Psi_n} \end{pmatrix} \tag{4.3}$$

such that for each $\boldsymbol{i}$ we can write $\boldsymbol{f_i^+} = \boldsymbol{\Psi_i}\boldsymbol{F^-}$. Finally the EoM (4.1) writes:

$$\boldsymbol{M_i(q_i)\ddot{q}_i} + \boldsymbol{C_i(q_i, \dot{q}_i)} + \boldsymbol{G_i(q_i)} = \boldsymbol{J_{i,0}^T f_i^0} + \boldsymbol{J_{i,-}^T f_i^-} - \boldsymbol{J_{i,+}^T \Psi_i F^-} + \boldsymbol{S_i \tau_i} \tag{4.4}$$

where $\boldsymbol{J_{i,0}}$ and $\boldsymbol{J_{i,-}}$ and $\boldsymbol{J_{i,+}}$ are the matrices obtained by extracting from $\boldsymbol{J_i}$ the columns corresponding to the positions of $\boldsymbol{f^0}$, $\boldsymbol{f^-}$, $\boldsymbol{f^+}$ in $\boldsymbol{f}$, respectively. Then all the Equations (4.4) are simply stacked in the compact form

$$\boldsymbol{M(q)\ddot{q}} + \boldsymbol{C(q, \dot{q})} + \boldsymbol{G(q)} = \boldsymbol{J_0^T F^0} + \left(\boldsymbol{J_-} - \boldsymbol{\Psi^T J_+}\right)^{\boldsymbol{T}} \boldsymbol{F^-} + \boldsymbol{S\tau} \tag{4.5}$$

with the following matrices and vectors

$$\boldsymbol{q} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_n}), \tag{4.6}$$

$$\boldsymbol{\tau} = (\boldsymbol{\tau_1}, \ldots, \boldsymbol{\tau_n}), \tag{4.7}$$

$$\boldsymbol{M(q)} = \mathbf{blockdiag}(\boldsymbol{M_1(q_1)}, \ldots, \boldsymbol{M_n(q_n)}), \tag{4.8}$$

$$\boldsymbol{J_0(q)} = \mathbf{blockdiag}(\boldsymbol{J_{1,0}(q_1)}, \ldots, \boldsymbol{J_{n,0}(q_n)}), \tag{4.9}$$

$$\boldsymbol{J_+(q)} = \mathbf{blockdiag}(\boldsymbol{J_{1,+}(q_1)}, \ldots, \boldsymbol{J_{n,+}(q_n)}), \tag{4.10}$$

$$\boldsymbol{J_-(q)} = \mathbf{blockdiag}(\boldsymbol{J_{1,-}(q_1)}, \ldots, \boldsymbol{J_{n,-}(q_n)}), \tag{4.11}$$

$$\boldsymbol{S} = \mathbf{blockdiag}(\boldsymbol{S_1}, \ldots, \boldsymbol{S_n}), \tag{4.12}$$

$$\boldsymbol{C(q, \dot{q})} = \begin{pmatrix} \boldsymbol{C_1(q_1, \dot{q}_1)^T} & \cdots & \boldsymbol{C_n(q_n, \dot{q}_n)^T} \end{pmatrix}^{\boldsymbol{T}}, \tag{4.13}$$

$$\boldsymbol{G(q)} = \begin{pmatrix} \boldsymbol{G_1(q_1)^T} & \cdots & \boldsymbol{G_n(q_n)^T} \end{pmatrix}^{\boldsymbol{T}}. \tag{4.14}$$

The kinematic constraint that expresses the coincidence of the contacts points corresponding to an action/reaction pair can be synthetically written using the matrix $\boldsymbol{\Psi}$ and the principle of virtual work as

$$\boldsymbol{J_+\dot{q} = \Psi J_-\dot{q}} \tag{4.15}$$

which is equivalent to, given that a permutation matrix is orthogonal $\boldsymbol{\Psi^T\Psi = I_{3K}}$,

$$\boldsymbol{\left(J_- - \Psi^T J_+\right)\dot{q} = 0} \tag{4.16}$$

This constraint has to be complemented with the static environment contact kinematic constraint that writes

$$\boldsymbol{J_0\dot{q} = 0} \tag{4.17}$$

The constraints and tasks expression as described in Section 2.2 remains exactly the same and we finally obtain a multi-robot system integrated in a single QP problem that is solved for each control iteration at once.

In the following sections of this chapter, we present and discuss experimental results on vision based motion control for complex tasks and articulated object configuration estimation for closed-loop manipulation.

4.2 Description of the experimental contexts

As mentioned in the Introduction, robots are now sharing work and home space with humans. Ongoing research efforts aim to enable robots to perform various tasks, like everyday human-scale manipulation activities or manufacturing tasks, previously done mostly by humans, at a competent level.

We have considered two different experimental contexts to assess the methodology built in this thesis. Both experimental contexts involve the humanoid robot HRP-4 and use the MQP control framework:

1. The circuit breaker experiment: the HRP-4 operate circuit breakers (switches) disposed on a panel.
2. The printer's drawer opening: the HRP-4 opens and closes a drawer of a printer.

4.2.1 The context of the circuit breaker experiment

The circuit breaker experiment takes place in the context of the collaboration between author's hosting group at CNRS-LIRMM and the Airbus Group aiming at

introducing humanoid technologies into the aircraft manufacturing. The motivation behind using humanoid platform is not solely driven by a wish to increase the level of automation, but also to use humanoids to perform "non-added value tasks": repetitive tasks, where the experience and intelligence of the operator are not put to use, and where boredom can lead to mistakes, or tasks presenting health risks, that are usually performed in confined spaces. For example,

- Accurate assembly operations, e.g. riveting, drilling and screwing using manual or semi-automatic hand tools;
- Cleaning and painting operations;
- System installation: electrical harness installation, connector plugging, etc.

Humanoids could also be used to perform other non-added value tasks such as conveying equipment or tools to highly qualified operators. By unburdening highly qualified operators from such boring tasks, a robot (even a costly humanoid) is socially and economically valuable. In such a context, the 3D models of the aircraft and the shop floors exist and are constantly updated in the manufacturing process, enabling model-based reference and localization of the robot.

Our case study covers a frequent use-case encountered in production. It aims at investigating the capability (and hence the feasibility) of a humanoid to check the correct behavior of the electrical systems of the plane or helicopter once it has been assembled. Circuit breakers are used on aircrafts in order to protect an electrical circuit from damage caused by current excess. Their state can also be switched manually in order to reset operation or to switch off a function. In particular, during ground testing, operators manipulate the circuit breakers panel in order to validate the behavior of the systems in a large number of configurations. This is done in the context of the COMANOID[1] EU project. Here, we demonstrate how the HRP-4 controlled by the MQP and equipped with visual feedback can perform the task of checking if the circuit breaker switches have been properly installed by pulling them with specially designed tool attached to the hand of the humanoid.

4.2.2 The context of opening the drawer of a printer

Our second experimental context is in line with the RoboHow[2] European project, which is focused on the research to enable robots to perform manipulation activities, which human perform in their daily life. Enabling robots to perform everyday manipulation activities such as offices or household chores exceeds, in terms of task, activity, behavior and context complexity, anything that have been investigated in motion planning, cognitive robotics, autonomous robot control and

[1] `www.comanoid.eu`
[2] `www.robohow.eu`

artificial intelligence at large. It is not feasible to expect human programmers to equip our robots with plan libraries that cover such an open-ended task spectrum competently. The research lines taken under the RoboHow project is that a key enabler for meeting the "open world" challenge is to exploit the huge information resources that already exist and are intended for human use, such as web services, product catalogues, dictionaries and encyclopaedic knowledge bases... and bridge them with a sophisticated low-level controller, in which tasks specification is simplified.

Ideally, we favor a knowledge-enabled control by affording objects with more data than simple geometry and inertia parameters. For example, each object (e.g. drawer) may contain information concerning the parameters of the tracker and all what it needs to increase performances. This extends *constraint- and optimization-based movement specification* and execution methods, that permit the force adaptive control of movements, that achieve the desired effects and avoid the unwanted ones. In addition, novel perception mechanisms, such as the estimation of the articulated objects configuration and satisfying the knowledge preconditions of plans, allow monitoring the effects of actions that will make the whole approach more feasible.

The constraint- and optimization-based movement specification, that is best illustrated with our MQP control framework, builds a stable and sustainable bridge between symbolic high-level control and the continuous time and space of the robots' motion and perception. At the high level, the robot will reason about and manipulate symbolic descriptions of these movement specifications, while at the low level, the continuous time and space trajectories will have to satisfy the specifications. In fact the idea of the MQP is that the model already embeds the constraints (the kinematics and the dynamics ones) instead of explicitly defining them. This property substantially simplifies the tasks specifications.

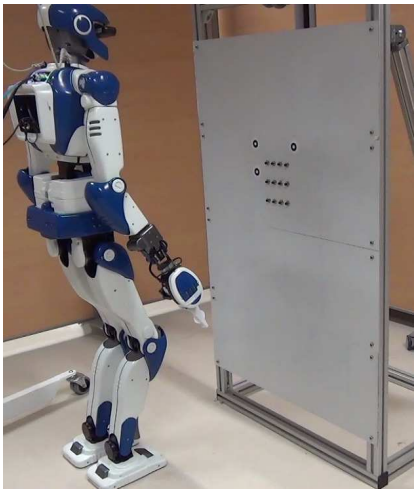
For the articulated object manipulation case study we consider the example of opening drawer of a printer, as in the RoboHow project, we had to demonstrate the HRP-4 operating in working or home offices.

4.3 The circuit breaker case study

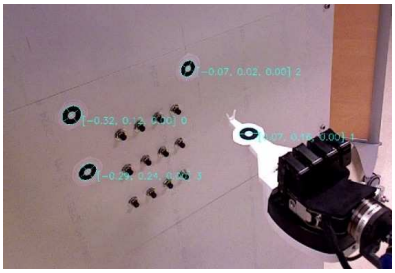
4.3.1 Description of the experimental setup and requirements

The robot used for the experiments is the humanoid HRP-4. An Xtion PRO LIVE camera is mounted in the head of the HRP-4. The robot is placed in front of the panel with the circuit breakers (provided by the Airbus Group). The initial position does not have to be precisely set, as visual feedback allows to perform the

task successfully regardless of the variability in the starting position. Figure 4.2a shows one example of a feasible initial placement.



(a) Example of initial position.



(b) Marker and tool placement.

Figure 4.2: Experimental setup.

Switches on the panel are given a label based on the position in the $\mathbf{3} \times \mathbf{4}$ grid, that is $\texttt{switch}_{ij}$, where $\boldsymbol{i}$ and $\boldsymbol{j}$ indicate the row and the column respectively.

Three WhyCon [69] markers are placed in the panel to form a pattern, which is not ambiguous in rotation (e.g. "L" shape). One marker is attached to the tool, such that it remains visible throughout the experiment. The pulling tool is attached to the right wrist of the robot. Example of tool and marker placement is shown in Figure 4.2b. The horizontal offsets from one of the markers on the panel to the $\texttt{switch}_{\mathbf{00}}$ are measured and later used to compute relative pose of the switch, which needs to be pulled, in the camera frame. Distance from the tool tip to the tool marker center $\boldsymbol{d}_{\text{tc}}$ is measured and later used to set appropriate error thresholds for visual servoing tasks.

As was mentioned in Section 2.6, task-aware contact planning was used to ameliorate operating circuit breaker task performance. The advantage of the humanoids is that they can use free end-effectors to support the main task execution. In this example, additional QP tasks for the left hand were implemented in order to make the HRP-4 put the palm of its left hand on the panel. This created additional closed kinematic chains (in addition to the existing ones due to feet contacts), which resulted in the robot being able to apply higher forces on the switches, generated from internal torques. It also prevented the robot from tilting toward the panel during pulling, and increases the equilibrium area and robustness.

4.3.2 The MQP tasks description

The panel is considered as a fixed robot-base having multiple links, each with a prismatic joint: the switch system. In this subsection we present the tasks, which form the MQP objective function in the circuit breaker experiment controller implementation. As described previously in Section 2.2, each task is defined as an error in the sensory space and has weight and stiffness value. General overview of the tasks and corresponding values for weight and stiffness is presented in Table 4.1.

Task name	Description	Weight	Stiffness
robotComTask	Balance task	1000	5
torsoOriTask	Torso orientation	10	5
postureTask	Preferred robot configuration	10	0.1
lhSurfaceTask	LH movement control	up to 1000	3.5
rhToolTask	RH movement control	up to 100000	8
gazeTask	IBVS task	5	5
rhPbvsTask	PBVS RH task	20000	5
rhOriTask	RH orientation task	100	5
forceLeftHand	Target force for LH	1000	8
forceTool	Target force for the tool	1000	8
rhPosTask	Avoid moving RH	1000	2
lhPosTask	LH movement control	10000	2
rhPushTask	Move tool back	20000	12

Table 4.1: The MQP tasks with weight and stiffness values.

4.3.3 Scheduling tasks using a finite state machine

We mentioned in Section 2.6 that the addition, removal and change of priority of tasks and constraints is managed by a FSM. Figure 4.3 illustrates the FSM implementation for the circuit-breaker controller. In the remaining part of this subsection, states and transitions of implemented FSM are described in more detail.

Initial posture is a safety state: CoM and torso orientation tasks must converge, before ***t*1** can be triggered and further tasks are then added to the QP. In *hand to panel* state, position task is added to move left hand (LH) to a predefined waypoint. ***t*2** is triggered after position task converges. When in *left hand impedance*, position task for LH is removed. New contact constraint is added to ensure that left palm and panel are in contact. Impedance task with desired target force for the contact is added. ***t*3** is triggered after desired force is achieved.

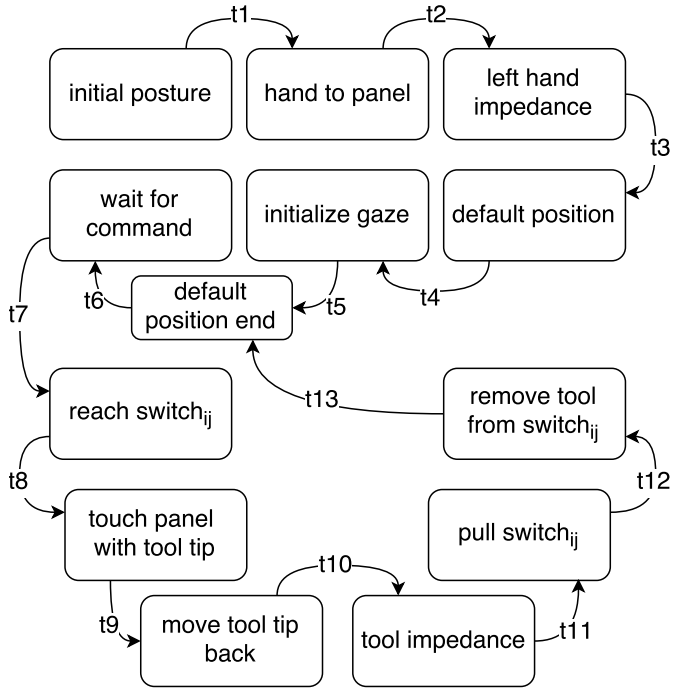


Figure 4.3: FSM of the circuit-breaker operating controller.

In *default position* state, position task is added to move the right hand (RH) to default position through two predefined way-points. $\boldsymbol{t4}$ is triggered once both waypoints have been reached sequentially. Image based visual servoing (IBVS) task is added in *initialize gaze* state, to ensure that markers appear in the FoV. Once IBVS error converges, $\boldsymbol{t5}$ is triggered. After entering *default position end*, position task for RH is removed. Default position of the right hand with respect to the panel is refined by adding position based visual servoing (PBVS) task. Once vision based position of the tool is less than **3**mm away from the target value, $\boldsymbol{t6}$ is triggered. In *wait for command*, robot is waiting for the command with switch label ($\boldsymbol{ij}$). After command is received, $\boldsymbol{t7}$ is triggered.

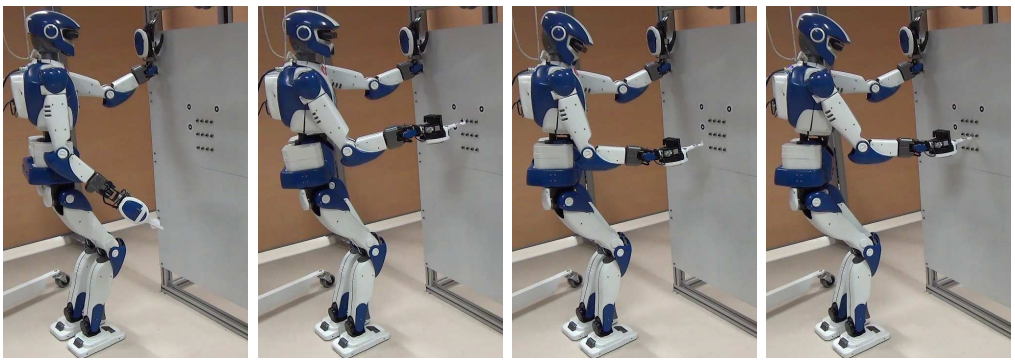
In *reach* $\texttt{switch}_{\boldsymbol{ij}}$, the target for PBVS task is redefined, so that the tip of the tool is positioned under $\texttt{switch}_{\boldsymbol{ij}}$ with appropriate offsets. $\boldsymbol{t8}$ is triggered once tool tip is at least **3**mm away from desired position. In the beginning of *touch panel with tool tip* PBVS task is removed. New force-guarded position task is added to detect when tool tip is in contact with the panel. When in contact (i.e. target force is reached), $\boldsymbol{t9}$ is triggered. Set point task with objective to move RH **5**mm back is added in *move tool tip back* state. $\boldsymbol{t10}$ is triggered once set point task error converges. In *tool impedance*, set point task is removed. New contact constraint and impedance task are added to bring tip of the tool in contact with $\texttt{switch}_{\boldsymbol{ij}}$. After impedance task converges, $\boldsymbol{t11}$ is triggered. In *pull* $\texttt{switch}_{\boldsymbol{ij}}$, set point task is added to move RH diagonally **1**cm up and **2.5**cm back to avoid tip of the tool slipping off the switch. As a result of this motion $\texttt{switch}_{\boldsymbol{ij}}$ is pulled. $\boldsymbol{t12}$ is triggered once Euclidean distance from the target position is less

than **8**mm. Contact between tool tip and the $\texttt{switch}_{ij}$ is removed in *remove tool from* $\texttt{switch}_{ij}$ state. After tool is removed away from the switch, $\boldsymbol{t}$**13** is triggered and robot goes back to *default position end* state.

4.3.4 Experimental results

Object parameter vector $\boldsymbol{q}$ (see Equation 3.1) is used in the MQP to model manipulated object. In order to perform manipulation tasks, elements of $\boldsymbol{q}$ need to be estimated, along with $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$, from the data of camera stream by means of computer vision. However, for the real-time motion control, which typically runs at the rate of 200 FPS, the estimation of the velocity and the acceleration from the camera frames is a known bottleneck, due to the fact that for the standard cameras frame acquisition rate is usually at most 60 FPS. In such setup, the required motion is executed and finished much before new command from the vision based motion control can be computed, which results in a jerky motion. Nevertheless, experiments with the MQP framework, which is an acceleration-resolved framework for motion control (i.e. acceleration of the actuated joints is a decision variable of the MQP), revealed that the great difference in frame rates of the low-level control and embedded camera frame acquisition do not cause significant jerkiness of the robot motions and allow to perform the tasks successfully.

For the discussion of the results, we indicate the main phases of the experiment as follows: 1) LH position task added, 2) LH impedance task added, 3) RH position task added, 4) IBVS task added, 5) PBVS task added, 6) switch label is received, 7) pulling finished, going back to default position. Brief illustration of the experiment state after some of the defined phases is shown in Figure 4.4. Figure 4.6 demonstrates changes in HRP-4 joint values at different stages of the



(a) After phase 2. (b) After phase 3. (c) After phase 5. (d) After phase 6.

Figure 4.4: Sample frames of the video recording of the experiment.

experiment. As can be seen, the joint value curves are smooth with no abrupt changes. Figures 4.7 and 4.8 illustrate the evolution of the task error values during the experiment. Description of the tasks with corresponding weight and stiffness

values is presented in Table 4.1. As can be seen from the plot, high weight CoM task error stays close to zero, while lower weight torso orientation task error varies. Figure 4.8 illustrates the fast convergence of IBVS, PBVS and LH position tasks to zero. Figure 4.5 illustrates the norm of main force components $\boldsymbol{f_x}$ of LH that is controlled to hold a given force on the panel, and RH force sensors (we plotted the norm of all components for RH). After phase 2, LH impedance task with force target **15**N is added and force value converges to this target. LH impedance task stays in the MQP until the end of the experiment, and the controller tries to satisfy the objective while other motions are being performed.

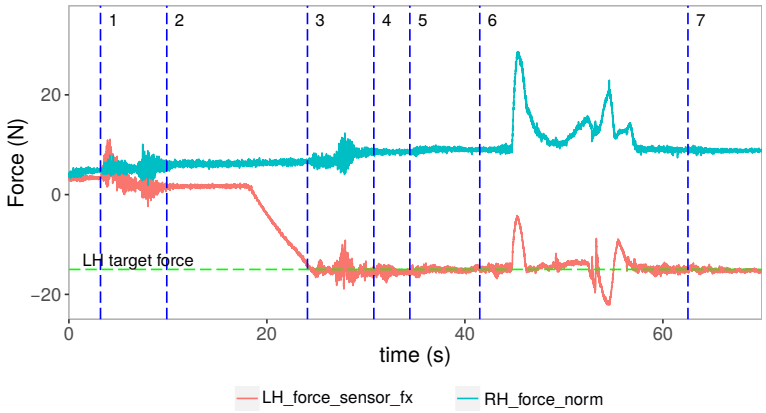


Figure 4.5: The LH is controlled in impedance after phase 2 to hold a contact of **15**N with the panel all along the experiment.

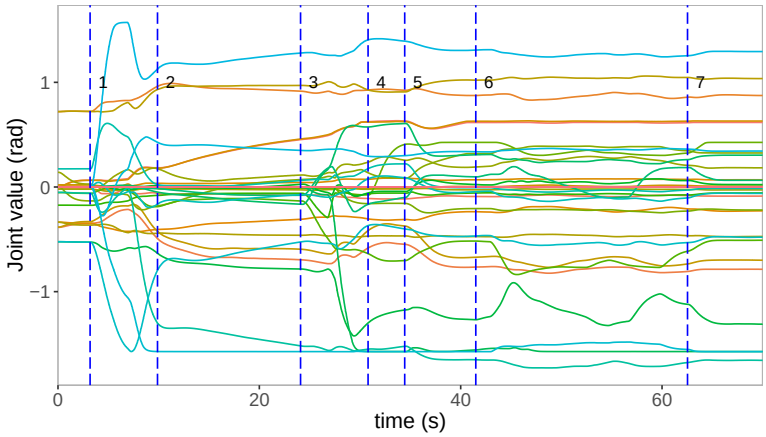


Figure 4.6: All HRP-4 joint values during the experiment.

Stiffness of the tasks can be increased to force faster convergence of the error. However, when using visual feedback, there is a limitation on task convergence speed, due to the low frame rate of the standard camera. Task stiffness values used in QP controller are presented in Table 4.1. Final size of the WhyCon markers used in the experiment, is **2.4**cm (diameter of the outer black circle). In order to

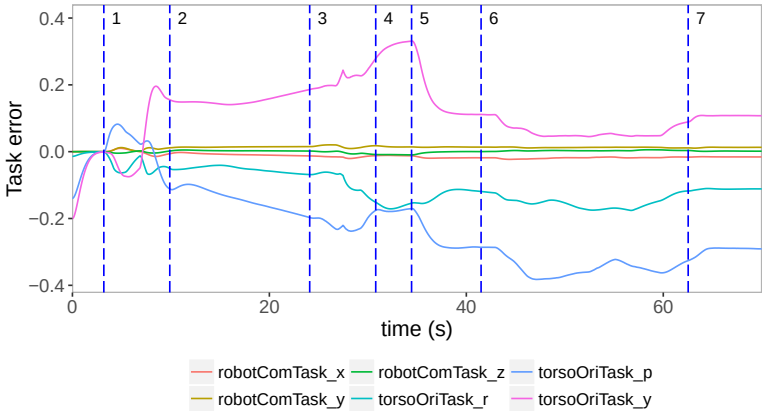


Figure 4.7: General MQP tasks: CoM and torso orientation task errors.

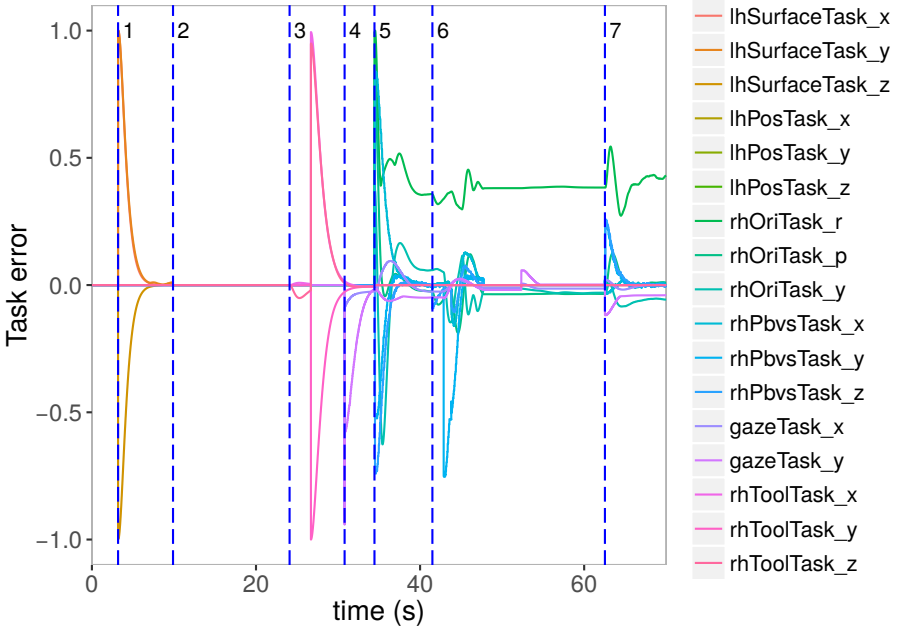


Figure 4.8: Experiment specific task errors (normalized).

minimize the amount of the potential error in the marker position estimation, the maximal camera resolution was used for frame acquisition ($\mathbf{1280 \times 1024}$ pixels). This allowed having $\mathbf{2-3}$mm precision in marker localization. Calibrated camera was used to ensure low uncertainty in the camera intrinsic parameters.

4.3.5 Conclusion

We have demonstrated that circuit breaker operation can be divided into states, to form an appropriate FSM, and implemented as a set of MQP tasks and constraints. This results in a unified force and vision based control, which enables

the HRP-4 to perform required operation. From the plots, which demonstrate the MQP controller state and the robot's sensor values during the experiments, we see that task errors decrease exponentially and force sensor values remain around the specified target values. The experiments also proved the importance of creating additional closed-kinematic chain to increase robot's equilibrium and pulling forces.

The experiments were successfully conducted several times in the laboratory over the course of two months without imposing any explicit constraints on the light conditions of the room (both natural and electric light were used at some point). The same controller, with some additional improvements in the implementation, was used to demonstrate this work at the first Digital Festival in Tahiti in March of 2017 (Figure 4.9). This shows, that our experiment is highly reproducible.

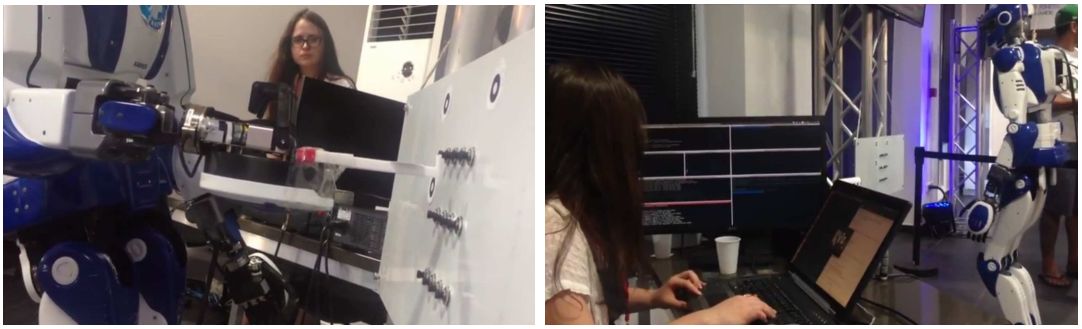


Figure 4.9: Demonstration of the Airbus experiment at the Digital Festival, Papeete, French Polynesia, March 2017.

4.4 The printer drawer case study

In this section, we present the results of the closed-loop experiments of the poly-articulated object manipulation, namely the printer drawer opening, with the HRP-4 robot controlled by the MQP framework. The estimation of the object configuration is done via visual feedback methodology described in Chapter 3.

4.4.1 Experimental setup

The humanoid robot HRP-4 is equipped with a Xtion PRO LIVE RGB-D sensor. The Xtion is used as monocular camera, providing images with a resolution of $\mathbf{640 \times 480}$ pixels at 30 FPS. A calibration procedure provides the intrinsic parameters used in the projection model of the algorithm.

For the initial experiments visual markers were used. This was done in order to eliminate potential visual point feature detection failures and to focus mainly on

the configuration reconstruction part of the VVS based poly-articulated object tracking methodology. The images are processed by WhyCon, a vision-based localization library that detects proper markers placed in the field-of-view of the camera, and also gives an estimate of their position in the camera frame. Four WhyCon markers were placed at the known positions on the FB of the printer and two more were placed on the drawer as shown in Figure 4.10.

Figure 4.10: Markers placement on the printer.

These markers represent the $\boldsymbol{p}$ PoI in our algorithm, and the detection of their corresponding visual features fills the vector $\boldsymbol{s}^*$. For the continuation of this work, the markers can be replaced with keypoint features such as SIFT, however, this would require additional efforts to address potential failures in point detection as well as speed-up implementation to allow real-time performance, as real-time closed-loop control is very sensitive to both of those factors.

In the start of the estimation experiment, the robot is placed in front of the printer with the hand grasping the drawer. Then, commands are sent to the robot with specification of the drawer opening amount. The robot is supposed to execute these commands. The same set-up is used for the experiment with edges, except that the visual markers are not put.

The configuration vector $\boldsymbol{q} \in \mathbb{R}^{\mathbf{9}}$ that we provide is composed of (i) the pose (position vector and quaternion) of the printer FB, (ii) the value of the prismatic joint of the drawer, $\boldsymbol{q_1}$, and (iii) the distance between the two markers on the drawer, $\boldsymbol{q_2}$, modelled as a virtual prismatic joint.

4.4.2 Integration of the poly-articulated object configuration estimator to the MQP controller

The HRP-4 robot is controlled with our MQP [68]. The tasks, which are defined in the controller for the printer's drawer experiment, are presented in the Table 4.2,

Task name	Description	Weight	Stiffness
robotComTask	Balance and CoM position task	1000	5
torsoOriTask	Torso orientation	500	5
postureTask	Preferred robot configuration	0.1	5
lhSurfaceTask	LH movement control	1000	2
rhPositionTask	RH movement control	100	0.1
rackTask	Drawer position task	1500	2
rackPostureTask	Drawer posture task	10	2

Table 4.2: The MQP tasks with weight and stiffness values.

with corresponding values for the weight and stiffness. In order to achieve the closed-loop behavior, it is possible to define the error between the current value of the drawer's joint and a desired joint target: $\boldsymbol{\tau_q} = (\boldsymbol{q_1} - \boldsymbol{q_{1,d}})$. Here, $\boldsymbol{q_{1,d}}$ is specified by the user (it could be defined by a higher level planning), while $\boldsymbol{q_1}$ is provided by our method. This term is actually added to the cost function of the MQP after the robot grasps the drawer

$$\boldsymbol{w_q} \left\| \ddot{\boldsymbol{\tau}}_q + 2\sqrt{\boldsymbol{k_p}}\dot{\boldsymbol{\tau}}_q + \boldsymbol{k_p}\boldsymbol{\tau_q} \right\| \tag{4.18}$$

with $\ddot{\boldsymbol{\tau}}_q = \boldsymbol{J_q}\ddot{\boldsymbol{q}}_1 + \dot{\boldsymbol{J}}_q\dot{\boldsymbol{q}}_1$, being $\boldsymbol{J_q}$ the Jacobian of the task; $\boldsymbol{w_q}$ is a given weight. The addition, editing and removal of the tasks in the MQP is handled by the FSM,

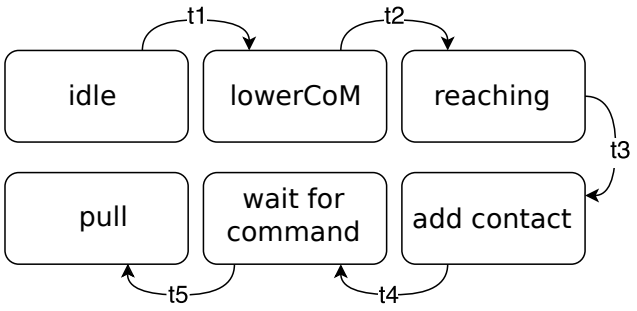


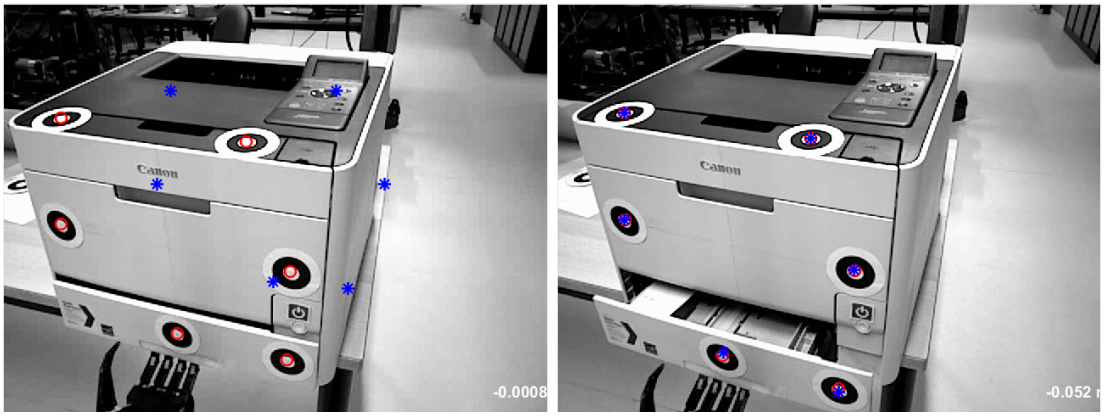
Figure 4.11: FSM for the drawer experiment.

similarly to what has been presented in Subsection 4.3.3. Figure 4.11 shows the diagram of the FSM for the printer drawer opening experiment.

4.4.3 Point feature experiments and results

For the point feature based experiments, the initial value of the configuration vector has been set to $\boldsymbol{q} = (\mathbf{0.0, 0.1, 0.6, 1.0, 0.0, 0.0, 0.0, 0.0, 0.1})^T$ at $\boldsymbol{t} = \mathbf{0}$. Figure 4.12a illustrates the tracker state at $\boldsymbol{t} = \mathbf{0}$. Figure 4.12b shows the state of the experiment after the tracker has converged, and the manipulation has been

performed. Real experiments have revealed, that the computation of the visual features derivative, $\dot{\boldsymbol{s}}^*$, was very noisy. As already stressed before in this work, significantly noisy input is not acceptable in the real-time low-level control. In particular, since the camera and the object did not move excessively during the execution of the experiment, we disabled the derivative action in (3.7).



(a) Initial state of $\boldsymbol{s}(\boldsymbol{q})$ in blue and $\boldsymbol{s}^*$ in red.

(b) State of image point features after tracker converged.

Figure 4.12: Projection of PoI onto the image plane based on current configuration estimation (blue) and the corresponding detected point features (red).

Figure 4.13 shows the norm of the VVS error. After a small time interval, required to make the visual features converge to their real counterparts, the error decreases exponentially and remains small with no extreme jumps, showing the effectiveness of the visual feature tracking. To validate our approach, we compare the results of

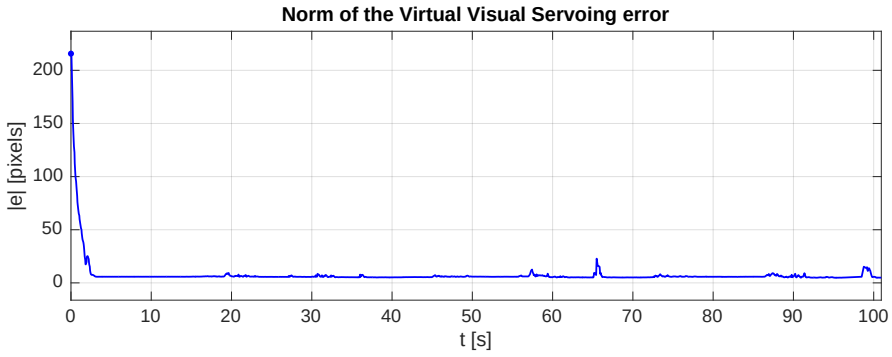


Figure 4.13: Experimental results: norm of the VVS error.

the proposed VVS-based tracking algorithm with a Singular Value Decomposition (SVD) based method for rigid motion reconstruction. It uses two sets of points: $\boldsymbol{m}$ PoI on the FB expressed in camera frame, ${}^c\boldsymbol{\rho}$, and the same points expressed in the object frame, ${}^o\boldsymbol{\rho}$. The first set is provided by WhyCon, the latter is assumed

to be known (i.e. manually measured). The reconstruction of the object pose (position vector $\boldsymbol{p_o}$ and rotation matrix $\boldsymbol{R_o}$) is formulated as a least squares error problem:

$$(\boldsymbol{R_o}, \boldsymbol{p_o}) = \underset{\boldsymbol{R_o}, \boldsymbol{p_o}}{\arg\min} \sum_{\boldsymbol{i}}^{\boldsymbol{m}} \|(\boldsymbol{R_o}\, {}^{\boldsymbol{o}}\boldsymbol{\rho_i} + \boldsymbol{p_o}) - {}^{\boldsymbol{c}}\boldsymbol{\rho_i}\|^{\boldsymbol{2}}. \tag{4.19}$$

To find an optimal combination of $\boldsymbol{R_o}$ and $\boldsymbol{p_o}$ that satisfies the minimization problem, we apply an SVD on the cross-covariance matrix of the two points distributions, that results in the decomposition $\boldsymbol{U\Sigma V^T}$. The sum in Equation (4.19) is minimized when $\boldsymbol{R_o} = \boldsymbol{VU^T}$. Then, the translation is computed as $\boldsymbol{p_o} = \boldsymbol{\mu_c} - \boldsymbol{R_o\mu_o}$, where $\boldsymbol{\mu_c}$ and $\boldsymbol{\mu_o}$ are the centroids of the sets ${}^{\boldsymbol{c}}\boldsymbol{\rho}$ and ${}^{\boldsymbol{o}}\boldsymbol{\rho}$, respectively. For the proof, interested reader can refer to [70]. Once transformation between camera and object frames is computed, FB parameters can be validated. For the validation of the drawer prismatic joint, we consider the 3D position of a marker on the drawer in $\boldsymbol{\mathcal{F}_o}$ reconstructed using $\boldsymbol{R_o}$ and $\boldsymbol{p_o}$. The amount of drawer opening is equal to the position of the drawer marker on the $\boldsymbol{x}$-axis (red) of $\boldsymbol{\mathcal{F}_o}$ (Figure 3.1c). For the validation of estimated distance between two markers on the drawer, Euclidean distance between corresponding points is computed using coordinates in $\boldsymbol{\mathcal{F}_c}$ provided by WhyCon.

The position of the FB as estimated by the VVS and by the SVD-based method is shown in the plots of Figure 4.14, by the blue continuous lines with triangle markers and red dashed line respectively. After an initial time interval, required to recover from the inaccurate initial value of $\boldsymbol{q}$, the estimation of the position of the FB, provided by the VVS, converge to the actual position of the object. The estimation curves provided by the SVD-based method validate the estimation results. Similarly, plots of the FB orientation, transformed from quaternion representation in roll-pitch-yaw angles for more intuitive representation, are presented in Figure 4.15. The effectiveness of the VVS at estimating the pose of the FB is validated by the comparison with SVD-based method. Furthermore, it appears to be less noisy and thus more appropriate to be used as a control feedback.

The plots in Figure 4.16a refer to the estimated value of the joint of the printer drawer. The black dash-dot line shows the command with desired joint values, sent to the MQP. Blue and red lines show drawer joint values estimated by the VVS and SVD-based methods respectively.

Finally, we show the estimation result of the distance between the markers placed on the drawer of the printer, modelled as a prismatic joint in our algorithm. As confirmed by the plot in Figure 4.16b, the method successfully estimated this distance, which represents the link length.

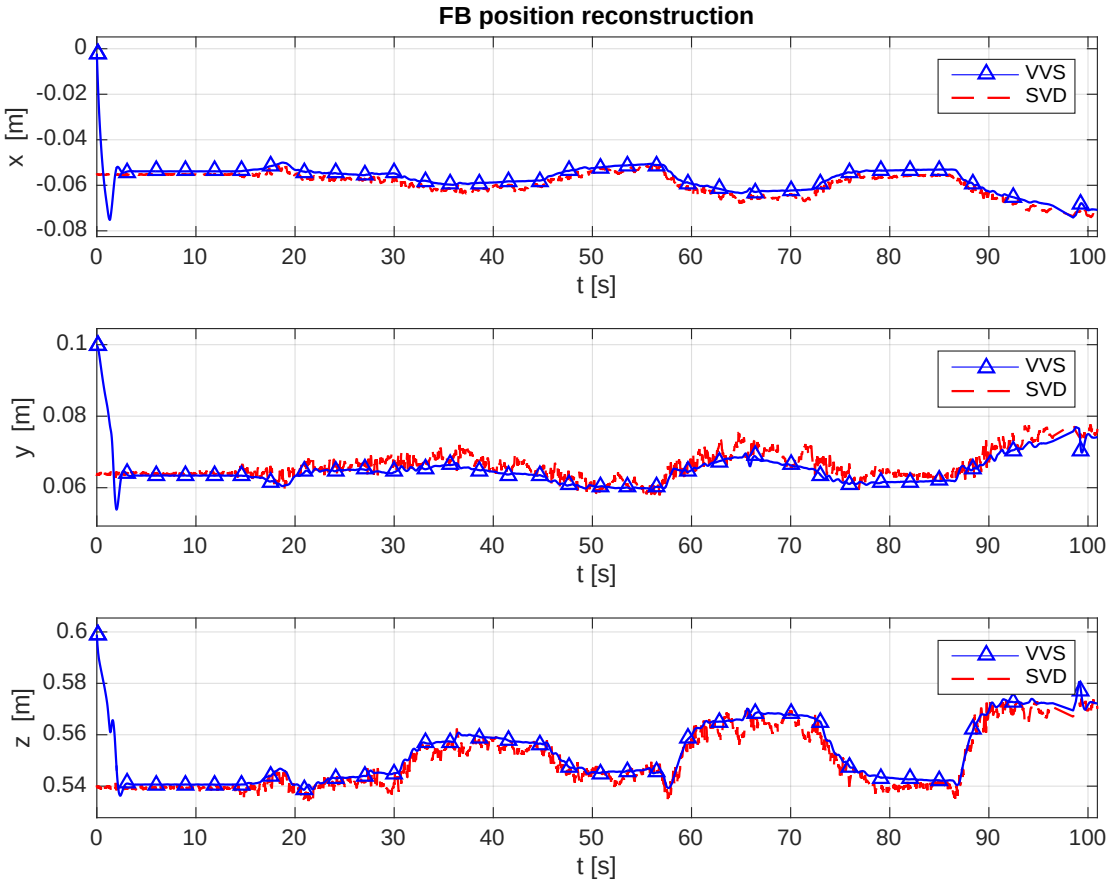


Figure 4.14: Experimental results: position of the object w.r.t. the camera frame. From top to bottom: $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$-coordinate.

4.4.4 Conclusion

We have shown, that articulated object tracking methodology, presented in Chapter 3, is applicable to the real-time control for robotic manipulation of the articulated objects in the MQP framework. Values of the configuration vector elements estimation have been validated by comparing estimation against the plots of SVD based method for computing position, orientation and drawer opening. Plots of the estimated values show that VVS based estimation of the articulated object configuration estimation is approximately equal to the values obtained using SVD. The curves of VVS based method have also proven to be more smooth. The geometric properties of the manipulated object, i.e. link size, have been also correctly estimated by proposed tracker. Experiments with the HRP-4 humanoid robot controlled by the MQP framework show that the robot can successfully perform the required manipulations.

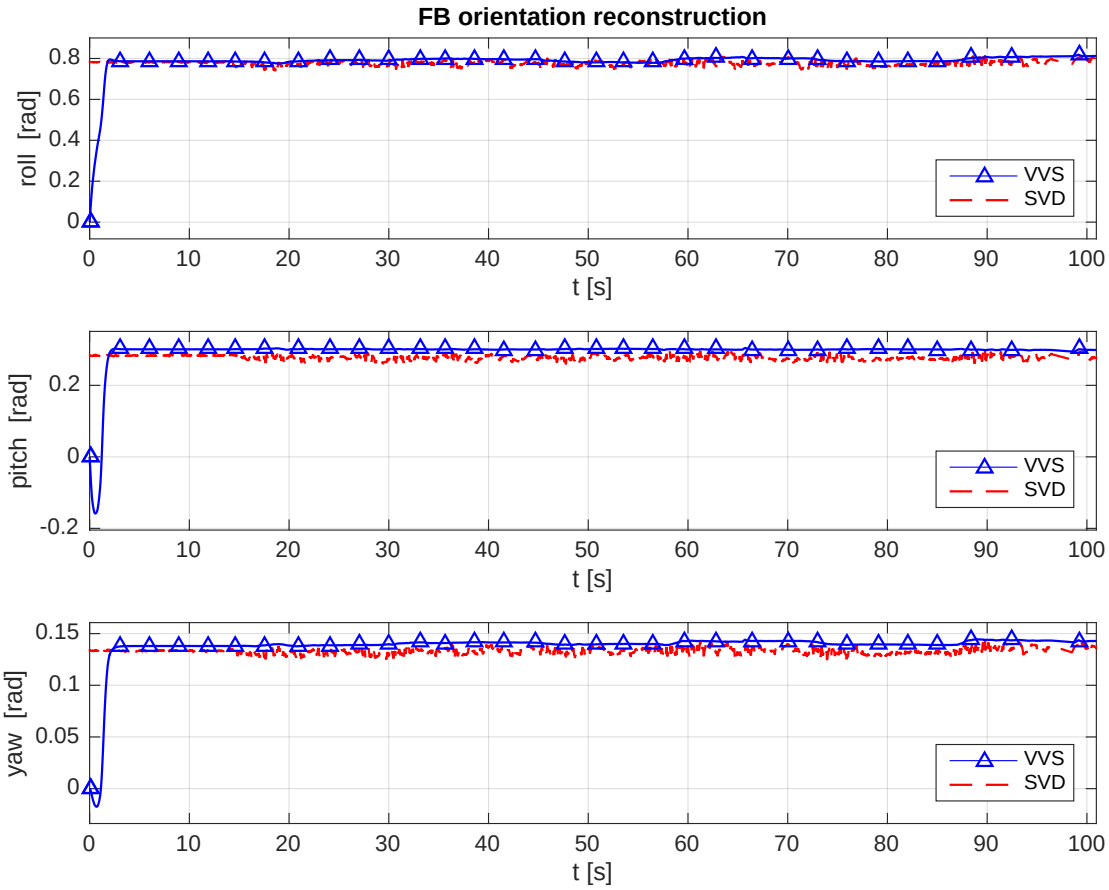
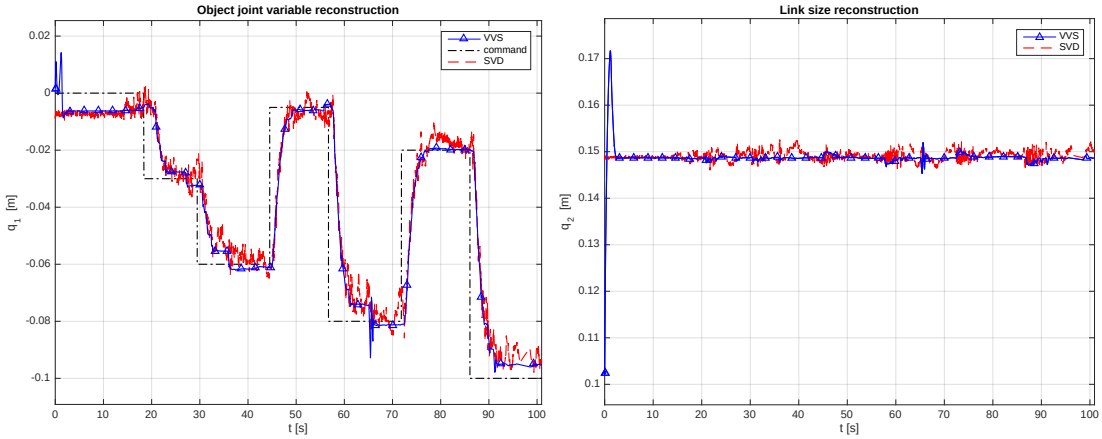


Figure 4.15: Experimental results: orientation of the object w.r.t. the camera frame. From top to bottom: roll, pitch and yaw angle.



(a) Experimental results: position of the drawer prismatic joint.

(b) Experimental results: distance between the markers of the drawer.

Figure 4.16: Plots for the drawer joint and size parameter estimation validation.

Conclusion

The aim of our work was to study visual feedback for robotic manipulation in the task-space multi-modal and multi-robot quadratic programming control framework (MQP). We devised and assessed a method to estimate the configuration of articulated objects that are not embedded with sensors. We reconstruct the configuration of articulated objected that are manipulated by a robot from the on-board camera data. The robot camera observes the object and the video images are processed in real-time. The processing consists in using the articulated object model and matching it to its counterpart in the video using virtual visual servoing (VVS) to close the control loop of the MQP controller.

In the conclusion, first, we review briefly our main contributions. Then, we outline current limitations and the short-term and mid-term efforts, required to overcome those limitations. Finally, we discuss the future directions of the research, which could result from the continuation of the presented work.

The overview of the *state-of-the-art* and experiments with available open source codes were presented in Chapter 1. The study of the *state-of-the-art* revealed the main challenges in vision based articulated object configuration reconstruction in the context of the low-level task-space robotics control. In the conclusion of the first chapter, the main challenges are outlined (Section 1.6). The remaining part of our work was devoted to account for the limitations and consider the challenges that need to be addressed, in order to enable robust visual feedback in robotic manipulation control.

The methodology for the motion control in the QP framework, including vision based motion control, has been described in Chapter 2. The integrated force and vision based control of the humanoid robot in the QP framework has been studied on a sample case of the Airbus circuit breaker panel. The implementation of the QP controller and the experimental results have been described in detail in Section 4.3. The experiments revealed the feasibility of achieving millimetre level precision of marker-based visual feedback for the meticulous manipulation of small switches of the circuit breaker panel. The concept of task-aware contact planning was used in the experiment, which resulted in increasing robot's equilibrium and maximum pulling forces.

The VVS based methodology for the reconstruction of the configuration of an

articulated object was presented in Chapter 3. First, the general framework was described for any kind of feature. Then, more detailed description of point and edge feature based configuration reconstruction of an articulated object through the means of VVS based method were described. The framework consisting of combined machine learning and guided Hough edge feature extraction was proposed for markerless robust, precise and real-time feature extraction. The experiments of the robotic manipulation of a sample articulated object (drawer opening), using the presented VVS based method for the visual feedback about the articulated object configuration, were conducted. The controller implementation details and experimental results for this part of the work, were presented in Section 4.4. The plots illustrating the estimated values of the object configuration vector elements show, that proposed configuration reconstruction method produces smooth and correct estimation of the object configuration. The plot of the value of the drawer's prismatic joint shows that robot, controlled by the MQP framework, could properly execute required manipulation task. We also discover that VVS can even be used to estimate the size and dimensions of the object if such dimension can be modelled as additional joints to those of the robot.

What concerns the limitation of our work in the current implementation of the QP controller for the circuit breaker experiment, only position of the hand tool is controlled. That causes a problem, that for the successful execution of the task, robot needs to be placed aligned more or less in parallel to the panel. In the next implementation of the controller, the orientation of the hand tool will also be controlled, however that requires to enable more robust orientation estimation of the tool. Currently, single WhyCon marker is used for the visual feedback about the tool. A single WhyCon marker only provides 2D orientation, which is not very reliable. Another type of markers can be considerer or more advanced solution using depth information could be used to estimate the tool orientation. For the panel position and orientation a template matching could be considered. That would allow to remove the markers from the panel and test the feasibility of using template matching for visual feedback for the control in complex setting, where frame-to-frame consistency and millimetre level accuracy are crucial for the successful task execution.

For the current version of the work, we took advantage of the visual markers to test our methodology without imposing any risk to the robotic manipulator, HRP-4, during trial manipulation experiments. Now that the configuration reconstruction method itself has been tested and validated, the continuation of this work will focus on replacing visual markers with highly robust edge or point feature detectors. The main problem that we faced, was that existing point feature detectors, such as SIFT, SURF, ORB... do not provide sufficient accuracy and frame-to-frame consistency for the use as a visual feedback for the low-level control. To test markerless feature based configuration reconstruction in real manipulation experiments, efforts will be dedicated to the implementation and meticulous testing of the proposed methodology for markerless feature extraction As a mid-term effort

for the enhancement of the proposed tracking methodology, one could also investigate properties of the *articulation matrix*, $\boldsymbol{A}$, for possible speed-up of the inversion operation. As this matrix often consists of several similar Jacobians, it could be possible to use known form of $\boldsymbol{A}$ and it's properties to increase computational performances.

An important long-term goal, that would enhance the presented tracking methodology, is to derive the control-law of the tracking, so as to express the acceleration of the articulated object configuration vector, $\ddot{\boldsymbol{q}}$, which is already part of the decision variables of the MQP control formulation. When such an implementation is achieved, the estimation and the control will be computed in parallel in the same framework. Also, since the articulated object is a part of the MQP, the Jacobians for the estimator, since already computed for the control part, can simply be reused. Moreover, since the constraints on the joints limits are also part of the control, the estimation can be integrated with fewer computation time and benefit from theses constraints. Yet, the implementation and testing of such an extension of our tracking methodology will also require more study on the acceleration estimation from vision.

The long-term research objectives, which will be developed as a continuation of the work conducted in the scope of this thesis, will be focused on extending the VVS based approach for the visual feedback of more complex articulated structures. For instance, using known and highly visible and distinctive visual features of the HRP-4 humanoid model, the effectiveness of the proposed tracker can be validated for the use in full humanoid robot configuration estimation. The important note here is that, unlike passive daily articulated objects, the HRP-4 has encoders, which allow to get a reliable ground truth for our method evaluation and validation. After visual feature extraction for the HRP-4 case is defined and implemented, the experiments will reveal, if any additional challenges need to be solved if the methodology is to be extended to more complex structures. The further continuation of the work will be dedicated to extending our approach for the human tracking, for the use in physical human robot interaction scenarios, such as physical human assistance. In this case, the type of visual features and the visual feature detection methods will change drastically, as humans have very various appearances. That would possibly requires to study completely different types of features to consider for the use in tracking. For example, dense depth information can be used in that case. Alternatively or additionally, deep learning based feature detection can be studied for human configuration estimation, with consideration, that result of such detection needs to be used in the low-level control for robotic manipulation. In this case, the main challenge is to ensure, frame-to-frame consistency and zero false-positive rate of visual feature detection. As deep learning approaches are usually probabilistic and non-deterministic models, overcoming this challenge, and ensuring safe and robust physical human robot interaction, is definitely a very challenging task.

Bibliography

[1] Q. V. Le, A. Saxena, and A. Y. Ng, "Active perception: Interactive manipulation for improving object detection," *Standford University Journal*, 2008.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. I–511, 2001.

[3] A. Treptow and A. Zell, "Real-time object tracking for soccer-robots without color information," *Robotics and Autonomous Systems*, vol. 48, no. 1, pp. 41–48, 2004.

[4] F. C. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH*, vol. 18, no. 3, pp. 207–212, 1984.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[6] D. G. Lowe, "Object recognition from local scale-invariant features," in *IEEE international conference on Computer vision*, vol. 2, pp. 1150–1157, 1999.

[7] M. Isard and A. Blake, "Condensation—conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[8] T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze, "Blort-the blocks world robotic vision toolbox," in *ICRA Workshop: Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[10] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision*, pp. 404–417, Springer, 2006.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *International conference on computer vision*, pp. 2564–2571, IEEE, 2011.

[12] K. Pauwels and D. Kragic, "Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking," in *EEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1300–1307, 2015.

[13] N. Kyriazis and A. Argyros, "Scalable 3d tracking of multiple interacting objects," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3430–3437, 2014.

[14] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints," in *IEEE International Conference on Computer Vision*, pp. 2088–2095, 2011.

[15] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Efficient model-based 3d tracking of hand articulations using kinect," in *British Machine Vision Conference*, (Crete, Greece), p. 3, 29 August 2011.

[16] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Tracking the articulated motion of two strongly interacting hands," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Crete, Greece), pp. 1862–1869, 16 June 2012.

[17] G. Park, A. Argyros, and W. Woo, "Efficient 3D hand tracking in articulation subspaces for the manipulation of virtual objects," in *Computer Graphics International*, (Crete, Greece), pp. 33–36, 28 June 28 – 1 July 2016.

[18] T.-H. Pham, A. Kheddar, A. Qammaz, and A. A. Argyros, "Towards force sensing from vision: Observing hand-object interactions to infer manipulation forces," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2810–2819, 2015.

[19] T.-H. Pham, N. Kyriazis, A. Argyros, and A. Kheddar, "Hand-object contact force estimation from markerless visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, submitted 2016.

[20] C. Choi and H. I. Christensen, "Robust 3d visual tracking using particle filtering on the special euclidean group: A combined approach of keypoint and edge features," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 498–519, 2012.

[21] C. Choi and H. I. Christensen, "3d textureless object detection and tracking: An edge-based approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3877–3884, 2012.

[22] M.-Y. Liu, O. Tuzel, A. Veeraraghavan, and R. Chellappa, "Fast directional chamfer matching," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 1696–1703, 2010.

[23] X. Tang, J. Su, F. Zhao, J. Zhou, and P. Wei, "Particle filter track-before-detect implementation on gpu," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, pp. 1–9, 2013.

[24] J. A. Brown and D. W. Capson, "A framework for 3d model-based visual tracking using a gpu-accelerated particle filter," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 1, pp. 68–80, 2012.

[25] G. Klein and D. W. Murray, "Full-3d edge tracking with a particle filter.," in *BMVC*, pp. 1119–1128, 2006.

[26] M. A. Goodrum, M. J. Trotter, A. Aksel, S. T. Acton, and K. Skadron, "Parallelization of particle filter algorithms," in *International Symposium on Computer Architecture*, pp. 139–149, 2010.

[27] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, "Real-time markerless tracking for augmented reality: the virtual visual servoing framework," *IEEE Transactions on visualization and computer graphics*, vol. 12, no. 4, pp. 615–628, 2006.

[28] P. Bouthemy, "A maximum likelihood framework for determining moving edges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 5, pp. 499–511, 1989.

[29] T. Schmidt, R. Newcombe, and D. Fox, "DART: Dense articulated real-time tracking," in *Robotics: Science and Systems*, (Berkeley, USA), 12 July 2014.

[30] C. Y. Ren, V. Prisacariu, O. Kaehler, I. Reid, and D. Murray, "3d tracking of multiple objects with identical appearance using rgb-d input," in *IEEE International Conference on 3D Vision*, vol. 1, pp. 47–54, 2014.

[31] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*, pp. 105–116, Springer, 1978.

[32] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 932–946, 2002.

[33] D. G. Lowe *et al.*, "Fitting parameterized three-dimensional models to images," *IEEE transactions on pattern analysis and machine intelligence*, vol. 13, no. 5, pp. 441–450, 1991.

[34] A. I. Comport, É. Marchand, and F. Chaumette, "Kinematic sets for real-time robust articulated object tracking," *Image and Vision Computing*, vol. 25, no. 3, pp. 374–391, 2007.

[35] É. Marchand and F. Chaumette, "Virtual visual servoing: a framework for real-time augmented reality," *Computer Graphics Forum*, vol. 21, no. 3, pp. 289–297, 2002.

[36] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[37] F. Chaumette and S. Hutchinson, "Visual servo control, part ii: Advanced approaches," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.

[38] D. J. Agravante, G. Claudio, F. Spindler, and F. Chaumette, "Visual servoing in an optimization framework for the whole-body control of humanoid robots," *IEEE Robotics and Automation Letters*, 2017.

[39] C. Samson, M. Le Borgne, and B. Espiau, *Robot control– the task function approach*. Oxford, England: Clarendon Press, 1991.

[40] L. Sentis, J. Park, and O. Khatib, "Compliant control of mult-icontact and center-of-mass behaviors in humanoid robots," *IEEE Transactions on Robotics*, 2010.

[41] J. Salini, S. Barthélemy, and P. Bidaud, *LQP-based controller design for humanoid Whole-body motion*, pp. 177–184. Springer, 2010.

[42] K. Bouyarmane and A. Kheddar, "Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (San Fransico, CA), pp. 4414–4419, 25-30 September 2011.

[43] L. Righetti and S. Schaal, "Quadratic programming for inverse dynamics with optimal distribution of contact forces," in *IEEE-RAS International Conference on Humanoid Robots*, (Businness Inoovation Center, Osaka, Japan), November 2012.

[44] P. M. Wensing and D. E. Orin, "Generation of dynamic humanoid behaviors through task-space control with conic optimization," in *IEEE International Conference on Robotics and Automation*, (Karlsruhe, Germany), pp. 3088–3094, May 2013.

[45] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[46] S. Kuindersma, F. Permenter, and R. Tedrake, "An efficiently solvable quadratic program for stabilizing dynamic locomotion," in *IEEE International Conference on Robotics and Automation*, (Hong Kong, China), 2014.

[47] J. Vaillant, A. Kheddar, H. Audren, F. Keith, S. Brossette, A. Escande, K. Bouyarmane, K. Kaneko, M. Morisawa, P. Gergondet, E. Yoshida, S. Kajita, and F. Kanehiro, "Multi-contact vertical ladder climbing with an HRP-2 humanoid," *Autonomous Robots*, vol. 40, no. 3, pp. 561–580, 2016.

[48] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, "A strictly convex hull for computing proximity distances with continuous gradient," *IEEE Transactions on Robotics*, vol. 30, pp. 666–678, 13 january 2014.

[49] S. Caron, Q.-C. Pham, and Y. Nakamura, "Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics," in *Robotics: Science and System*, 2015.

[50] J. Vaillant, K. Bouyarmane, and A. Kheddar, "Multi-character physical and behavioural interactions controller," *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[51] M. Nitsche, T. Krajník, P. Cizek, M. Mejail, and T. Duckett, "Whycon: an efficient, marker-based localization system," in *EEE/RSJ International Conference on Inteligent Robots and Systems*, (Hamburg, Germany), 2015.

[52] T. Schmidt, K. Hertkorn, R. Newcombe, Z. Marton, M. Suppa, and D. Fox, "Depth-based tracking with physical constraints for robot manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 119–126, 2015.

[53] M. Klingensmith, T. Galluzzo, C. Dellin, M. Kazemi, J. A. D. Bagnell, and N. Pollard , "Closed-loop servoing using real-time markerless arm tracking," in *International Conference on Robotics And Automation (Humanoids Workshop)*, May 2013.

[54] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3d object modeling," *The International Journal of Robotics Research*, vol. 30, no. 11, pp. 1311–1327, 2011.

[55] J. Deutscher and I. Reid, "Articulated body motion capture by stochastic search," *International Journal of Computer Vision*, vol. 61, no. 2, pp. 185–205, 2005.

[56] K. Nickels and S. Hutchinson, "Model-based tracking of complex articulated objects," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 28–36, Feb 2001.

[57] D. Katz and O. Brock, "Manipulating articulated objects with interactive perception," in *IEEE International Conference on Robotics and Automation*, (Amherst, Massachusetts, USA), pp. 272–277, 19 May 2008.

[58] R. Martín Martín and O. Brock, "Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2494–2501, 2014.

[59] R. Martín Martín, S. Höfer, and O. Brock, "An integrated approach to visual perception of articulated objects," in *IEEE International Conference on Robotics and Automation*, pp. 5091–5097, 2016.

[60] F. Zhao and B. G. M. van Wachem, "A novel quaternion integration approach for describing the behaviour of non-spherical particles," *Acta Mechanica*, vol. 224, no. 12, pp. 3091–3109, 2013.

[61] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[62] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*, vol. 73. Springer, 2011.

[63] P. E. Rybski, D. Huber, D. D. Morris, and R. Hoffman, "Visual classification of coarse vehicle orientation using histogram of oriented gradients features," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pp. 921–928, IEEE, 2010.

[64] O. Déniz, G. Bueno, J. Salido, and F. De la Torre, "Face recognition using histograms of oriented gradients," *Pattern Recognition Letters*, vol. 32, no. 12, pp. 1598–1603, 2011.

[65] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1867–1874, 2014.

[66] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics Springer, Berlin, 2001.

[67] J. Vaillant, K. Bouyarmane, and A. Kheddar, "Multi-character physical and behavioral interactions controller," *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[68] K. Bouyarmane, J. Vaillant, K. Chappellet, and A. Kheddar, "Multi-robot and force task-space control with quadratic programming," *IEEE Transactions on Robotics*, submitted 2017.

[69] M. Nitsche, T. Krajník, P. Čížek, M. Mejail, and T. Duckett, "WhyCon: an efficient, marker-based localization system," in *IEEE/RAS IROS Workshop on Open Source Aerial Robotics*, 2015.

[70] O. Sorkine, "Least-squares rigid motion using SVD," *Technical notes*, vol. 120, no. 3, p. 52, 2009.

License

Non-exclusive license to reproduce thesis and make thesis public

I, Anastasia Bolotnikova (date of birth: 12.12.1993),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   Articulated Object Tracking from Visual Sensory Data for Robotic Manipulation

   supervised by Gholamreza Anbarjafari and Abderrahamane Kheddar

2. am aware of the fact that the author retains these rights.

3. certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017