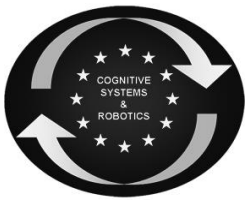
SAPHARI

SAFE AND AUTONOMOUS PHYSICAL HUMAN-AWARE ROBOT INTERACTION



Project funded by the European Community's 7th Framework Programme (FP7-ICT-2011-7)
Grant Agreement ICT–287513

Deliverable D4.1.1

Collision detection methods by means of sensor fusion

Deliverable due date: 31 October 2013	**Actual submission date: 31 October 2013**
Start date of project: 1 November 2011	**Duration: 48 months**
Lead beneficiary for this deliverable: IOSB	**Revision: 1.0**

Nature: R	**Dissemination Level: PU**
R = Report P = Prototype D = Demonstrator O = Other	PU = Public PP = Restricted to other programme participants (including the Commission Services) RE = Restricted to a group specified by the consortium (including the Commission Services) CO = Confidential, only for members of the consortium (including the Commission Services)

Executive Summary

This deliverable focuses on the close range monitoring, whose task is to supervise the close spatial range around the robot and the system evolution for a short-time horizon in order to identify sudden dangers. This includes both the contactless observation of the close robot environment as well as monitoring of direct physical contacts between human and robot.

At IOSB a 3D representation of the robot close range based on depth measurements has been developed. Firstly, the current close range is defined by computing a reachability grid. Then obstacles and occluded space are represented in an octree structure considering obstacle occlusions as well as robot occlusions. Based on the obstacle octree, the minimum distance between the robot and obstacles is determined.

SUN contributed to the methods described in this deliverable by working towards two objectives. Firstly, detecting collisions by means of a novel sensor, which combines proximity and contact information and secondly, exploiting distributed proximity information to create a safety shield around the robot that allows it to detect and react to unexpected collision situations.

UNIROMA1 has developed methods to monitor the intended and unintended physical contact between human and robot. The physical contact is detected and the resulting contact forces are estimated based on the robot joint positions without need for force-torque sensors. Based on a Kinect sensor, the contact point is located on the robot.

Table of contents

1 Introduction

Current robot manufacturing applications ensure safety by strictly separating humans from moving robots, which is achieved by fences or other physical barriers. If nevertheless a human enters the robot workspace, the robot is immediately stopped. These safety installations prohibit shared workspaces of humans and robots as well as physical human-robot interaction.

Therefore, robust monitoring concepts and components which enable the robot to detect as early and completely as possible dangerous situations in a weakly structured dynamic environment are developed in WP4. For better structuring, the surveillance problem is subdivided into close range monitoring and wide range monitoring.

Close range monitoring identifies sudden dangers within the actual working space of the robot. That implies both the supervision of a close spatial range as well as the observation of the system evolution only for a short-time horizon.

In contrast, wide range monitoring takes into account longer time intervals and consequently a wider spatial range. In this case the prediction of the future environment state plays a decisive role for a safe interaction.

This deliverable, that focuses on the close range monitoring task, contains work from IOSB, SUN and UNIROMA1. Different sensors are used to gain information on (possible) collisions and intended contact. Based on depth sensors, a 3D representation of the close robot environment is created to compute the minimum distance between the robot and obstacles. Using a sensor that combines proximity and contact information, collisions are detected. Based on distributed proximity sensors, a method for collision detection and avoidance is presented. Using only the measured joint positions, contact forces are estimated. The contact points are localized by means of a Kinect sensor.

2 Sensor placement

The work conducted on the problem of selecting and placing sensors for workspace monitoring in human-robot interaction by IOSB and UNIROMA1 was reported in Milestone MS13 – “Optimal placement and deployment of multiple sensors in large workspaces”.

First, different sensor categories were considered for the environment perception. The categorisation could then be used for a generalised sensor simulation to find an appropriate sensor selection and placement.

The sensor placement task was considered from two different points of view, distinguishing between the requirements of fixed manipulators and mobile manipulators.

Further details can be found in the MS13 report.

3 Acquisition of a 3D obstacle representation in the close range based on depth sensors

3.1 Problem formulation

To ensure safety in human-robot interaction, a 3D representation of the robot's close surroundings is a fundamental prerequisite. Based on the 3D environment representation, the minimum distance between robot and obstacles can be computed and collision detection and avoidance strategies can be performed. Therefore, the 3D representation has to meet the following requirements: The representation has to include static as well as dynamic obstacles. Furthermore, space occluded by the robot or by other objects has to be considered. To reduce the occluded and unobserved space, information from multiple sensors has to be fused. For the industrial SAPHARI scenarios, the method has to be applicable to mobile manipulators and sensors mounted on the robot.

3.2 Methodology

Due to limited computational resources, the detailed 3D representation of the robot environment is restricted to the robot's close range. To specify the close range, the space that is reachable by the robot within a certain time horizon is computed. Within the close range, the space occupied or occluded by obstacles is represented in an octree. Based on this octree, the minimum distance between the robot and the environment is determined.

Reachability grid

The robot's close range is defined here as the space that can be reached by at least one part of the robot within a certain time horizon based on the current robot state and the maximum velocities. To represent the reachable space, a so called reachability grid is computed.

In [1] a method is proposed to compute the reachability grid of a manipulator. For this method, each manipulator link is represented by a point cloud. Beginning with the end effector link, the link point cloud is rotated around the corresponding joint through the range of motion specified by the time horizon and the joint constraints. For each rotated point, the minimum time to reach this point is calculated. The resulting point cloud is reduced by collapsing it into a voxel grid. Then the reduced point cloud is attached to the next link point cloud and the former steps are repeated. Due to the voxelisation step at each joint the method is real-time capable even for high DoF manipulators.

To apply this approach to both mobile platform and manipulator, it is extended by translational motions similar to the described rotational motion. The omnidirectional platform is therefore modelled as one link, a rotational joint, and two prismatic joints for the translational movements. The considered motion constraints are the joint limits and the maximum joint and platform velocities. The acceleration constraints are disregarded, as the maximum accelerations of the robot are comparatively high. Furthermore, the error

due to the assumption of infinite acceleration leads only to an overestimate of the reachable space, which is uncritical for the close range definition.

As the method for the reachability computation treats all joint motions independently, it is not able to consider invalid robot configurations (e.g., due to self-collisions). But this results, like other simplifications, only in an overestimate of the reachable area.

The computational load can be further reduced, if the resulting grid contains only the information, if a cell is reachable or not, instead of the minimum time to reach the cell.

In Figure 1 the reachability map of the LWR manipulator based on the represented robot state is illustrated. The time horizon is set to 0.5s, the maximum joint velocities correspond to the manufacturer's data. The minimum time to reach a cell is encoded by colour.

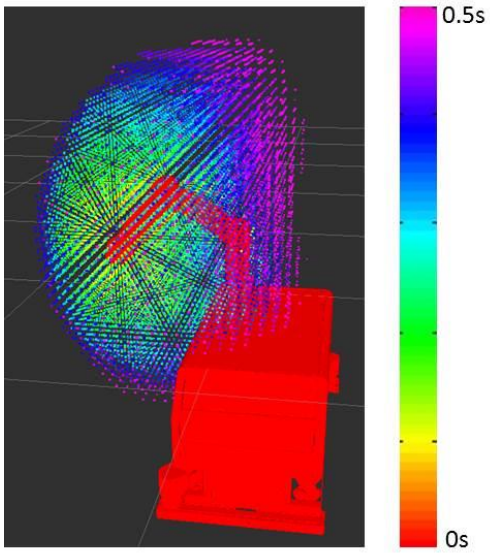


Figure 1 Reachability map of the robot arm with time horizon 0.5s

Obstacle octree

Based on multiple depth sensors' data a 3D representation of obstacles in the robot's close range is computed. For safety reasons the representation has to include obstacle occlusions, especially if an obstacle can occur in the space between the robot and the sensor.

Several approaches exist to model the 3D robot environment. One possibility is the use of octrees. An octree is a hierarchical data structure where each node represents a voxel that is subdivided into eight sub-voxels (nodes) till the minimum voxel size is reached [2]. As octrees allow for fast collision checks and distance computation e.g., by means of the *MoveIt* library [3], they are a suitable representation for obstacles in the robot close range. A further advantage is, that with the *OctoMap* library [2] a ready to use

open source framework exists that provides amongst others an octree data structure and ray tracing methods.

All kinds of depths sensors, as e.g., laser scanners, triangulation based sensors, or time-of-flight cameras, are modelled in a generalized way by a ray based model. The sensor model is described by a set of rays beginning in the sensor origin. The sensor measures the distance between the sensor and the point, where a ray hits the first object. In the following, these points are referred to as object points. The rays are specified by the opening angles and resolution of the sensor. The sensor field of view is defined by the opening angles and the maximum range.

It is assumed, that the above mentioned sensor properties opening angles, maximum range, and resolution are known. Furthermore the transformation between the sensor origin and the robot base frame is assumed to be given as well. In the case of a mobile platform with sensors mounted on the platform, the transformation between sensor and robot base frame is time invariant and determined by a calibration procedure. Otherwise, if the sensors are installed in the workspace, additionally, the robot pose has to be provided.

Based on the sensor properties for each sensor i the set $V_i(t)$ of all octree nodes that are located in the sensor's field of view is determined in each time-step t,

$$V_i(t) = \{n \epsilon T \mid n \text{ in the field of view of sensor } i\},$$

where T is an octree in the current robot base frame, limited to a certain volume expansion and n is an octree node. In the case of static transforms between sensor and robot base frame, V_i is time independent and has to be computed only once. But then, V_i must not be limited to the current robot close range, as the close range changes over time. Instead, the octree T is limited to a bounding box around the robot that is specified by the maximum dimensions of the robot close range over all possible robot states.

Before creating an obstacle representation is possible, the robot points have to be removed from the captured sensor data. For depth sensors that deliver a depth image this can be achieved by the Realtime URDF Filter from WP4.3 (SAPHARI Report Y1, [4]). The Realtime URDF Filter renders a virtual depth image based on the known robot model and the current robot state. Comparing the real depth image retrieved from the sensor with the virtual depth image allows for classifying all pixels into robot or other objects.

The retrieved depth data may be available in different formats, e.g., as depth image from a Kinect-like sensor, as a list of distance measures from a laser scanner or as point cloud. To obtain a sensor independent representation, the sensor data is first transformed into a robot and an obstacle point cloud in the sensor frame according to the classification result (e.g., by means of the Realtime URDF Filter). Even though the data is in the sensor frame, points that are located outside the octree's bounding box can be partly removed. Therefore, the maximum distance between the sensor origin and the bounding box edges is computed. All points with a depth measurement higher than this distance are certainly outside the

bounding box. Then the point clouds are transformed to the robot base frame and the remaining points outside the bounding box are filtered out.

Based on the pre-processed obstacle point clouds, for each sensor i the set $P_i(t)$ of all octree nodes that contain at least one obstacle point is determined. By means of ray tracing the nodes that are occupied or occluded by obstacle points $O_i(t)$ and the nodes occupied or occluded by robot points $R_i(t)$ are computed.

Then for each sensor the free space is given as the space in the field of view without the areas occupied or occluded by an obstacle or by the robot.

$$F_i(t) = V_i(t) \backslash (O_j(t) \cup R_j(t))$$

The free nodes $F_i(t)$ could also be directly computed by ray tracing between the sensor origin and all object points. But in this case, the object points outside the close range have to be considered as well. If $V_i(t) = V_i$ and if a significant amount of object points lies outside the close range, it is faster to compute the free nodes as proposed above.

Based on the reachability grid for the current joint state, all nodes that are reachable within a certain time horizon are included in the set $C(t)$.

The sensor data pre-processing and the computation of occupied and occluded nodes is done independently for each sensor in parallel processes. To obtain the final obstacle octree, the information about occupied and occluded nodes from all available sensors is fused according to

$$O(t) = \bigcup_i \left(C(t) \cap \left(P_i(t) \cup O_i(t) \backslash \left(\bigcup_{j, j \neq i} F_j(t) \right) \right) \right).$$

In the final octree all nodes are marked as occupied, that include a directly measured reachable obstacle point and all reachable nodes that are in the occluded space of one sensor and are not detected as free by another sensor. That means the monitored space is the sum of all sensor fields of view within the reachable area. Unsupervised space is not considered as obstacle, as the current sensor setup does not allow monitoring all space around the robot. Especially the space at the sides of the mobile platform is only surveyed by 2D laser scanners in one plane, as it is assumed that there are no overhanging objects at the side of the mobile platform that are not detected by the laser scanners. Similarly, space only occluded by the robot but not occluded by an obstacle is not considered as obstacle as well.

The proposed principle is illustrated in Figure 2 and Figure 3. In both examples, the left side shows the cells in the sensor field of view (green), cells occupied or occluded by an obstacle (red), and the cells occupied or occluded by the robot (blue) for two depth sensors. On the right side, the fusion result is shown, in which the red cells correspond to the occupied nodes in the final octree $O(t)$.

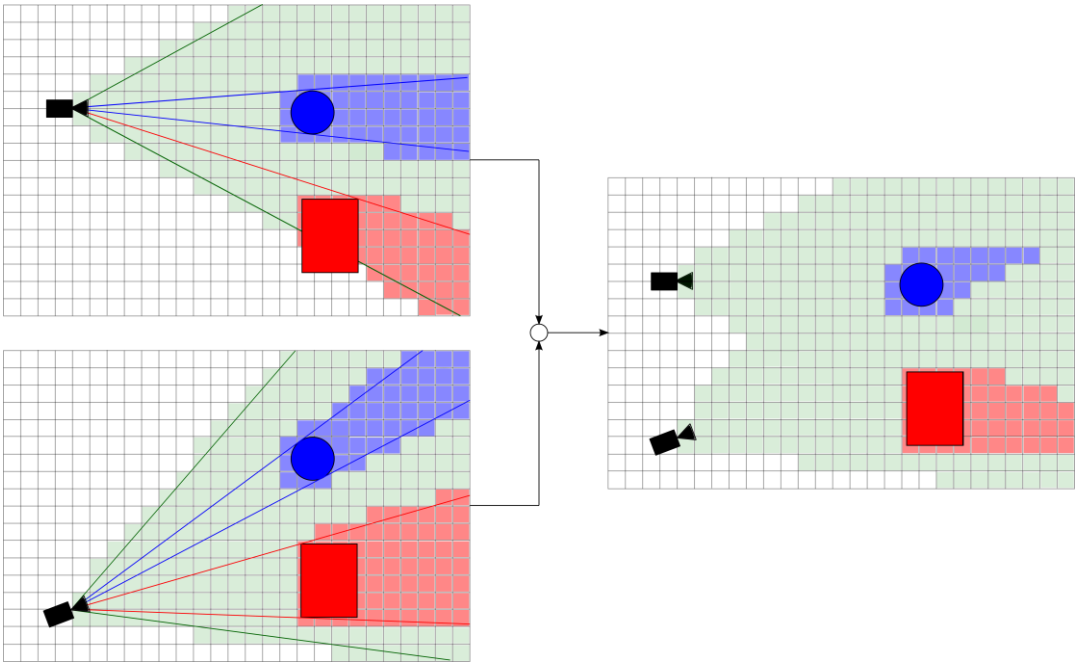


Figure 2 Illustration of the octree generation; left: field of view (green), robot occlusion (blue), and obstacle occlusion (red) of two depth sensors; right: fusion of both sensors

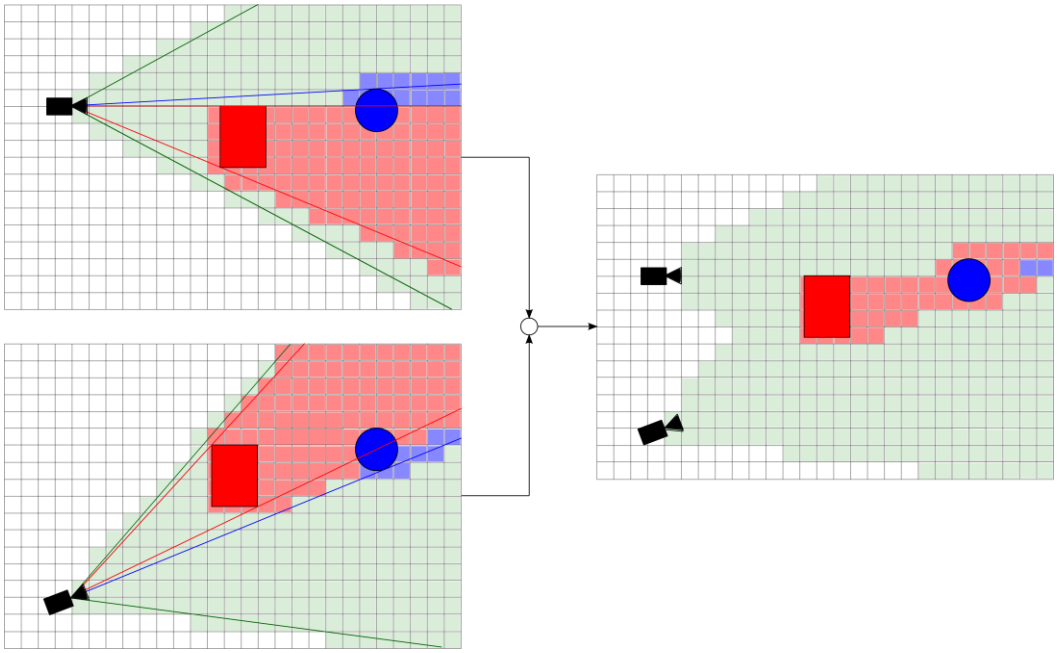


Figure 3 Illustration of the octree generation; left: field of view (green), robot occlusion (blue), and obstacle occlusion (red) of two depth sensors; right: fusion of both sensors

Minimum distance computation

Based on the resulting obstacle octree containing all nodes occupied or occluded by obstacles within the reachable area, the minimum distance between the robot and obstacles is computed by means of the open source library *MoveIt* [3].

3.3 Experimental setup and results

The proposed method has been applied to simulated sensor data and data acquired from real sensors on the robot. In both cases, the setup consists of the omnidirectional platform OmniRob equipped with a 7 DoF lightweight arm and four depth sensors mounted on the platform. Two Kinect sensors monitor the space around the robot arm. Their depth images are filtered by the Realtime URDF Filter to distinguish between robot and other objects. Two laser scanners observe the area on a horizontal plane around the platform near the ground floor.

For the extrinsic calibration of the 3D depth cameras relative to the coordinate system of the mobile platform, a novel calibration procedure has been developed which uses the robotic arm as a calibration target. This is motivated by the application: as the distance of objects to the arm is the main concern, the calibration between sensor and arm should be most accurate. By contrast, a conventional calibration pattern cannot be placed in the relevant part of the workspace because the arm will obstruct or occlude it, resulting in a lower accuracy of the calibration exactly where the highest accuracy is desired.

In our method, the 3D point cloud of the manipulator arm is acquired by the depth cameras. This point cloud is registered to an accurate 3D model of the arm using iterative closest point (ICP) methods. Figure 4 illustrates the registration of an acquired point cloud to the robot model.

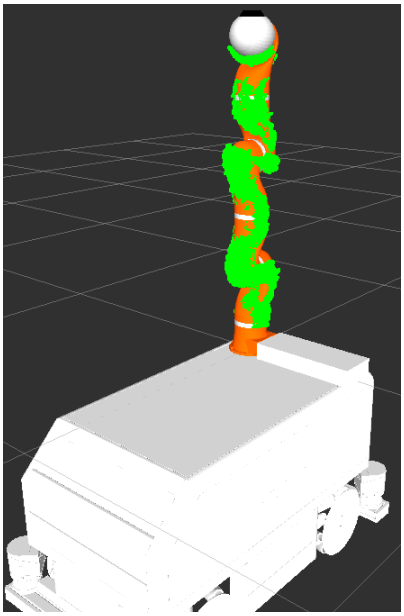


Figure 4 Sensor calibration. The acquired point cloud (green) is fitted to the robot model (orange/white).

In order to cover a wide field of view and to reduce ambiguities resulting from partial symmetry of the arm, point clouds of several different arm poses are incorporated in the registration process.

The whole data acquisition and registration procedure can be fully automated so that a frequent re-calibration is possible without human intervention.

Unlike a conventional calibration method used for comparison, the proposed approach achieves the accuracy required for the Realtime URDF Filter to remove exactly the robot points from the point cloud.

Figure 5 shows a simulated static scenario used for evaluating the 3D obstacle representation and the distance computation. In the left image of Figure 6 the resulting reachability grid of platform and arm for the illustrated robot state can be seen. The reachability grid is computed with a time horizon of 0.5s and a grid resolution of 0.09m. On a 2.8GHz Intel Core i7 quad-core processor running the simulation and octree generation in parallel, the reachability grid is updated with a mean frequency of 12Hz. The right image of Figure 6 shows the resulting obstacle octree. For better understanding, the occupied nodes are visualised red, the occluded nodes orange.

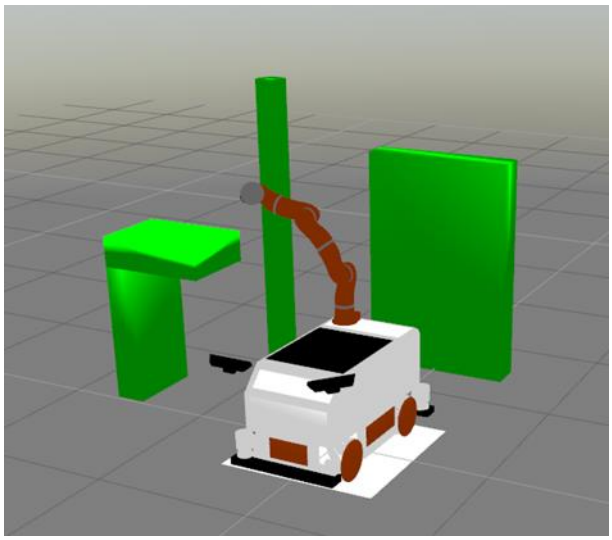


Figure 5 Simulated scene

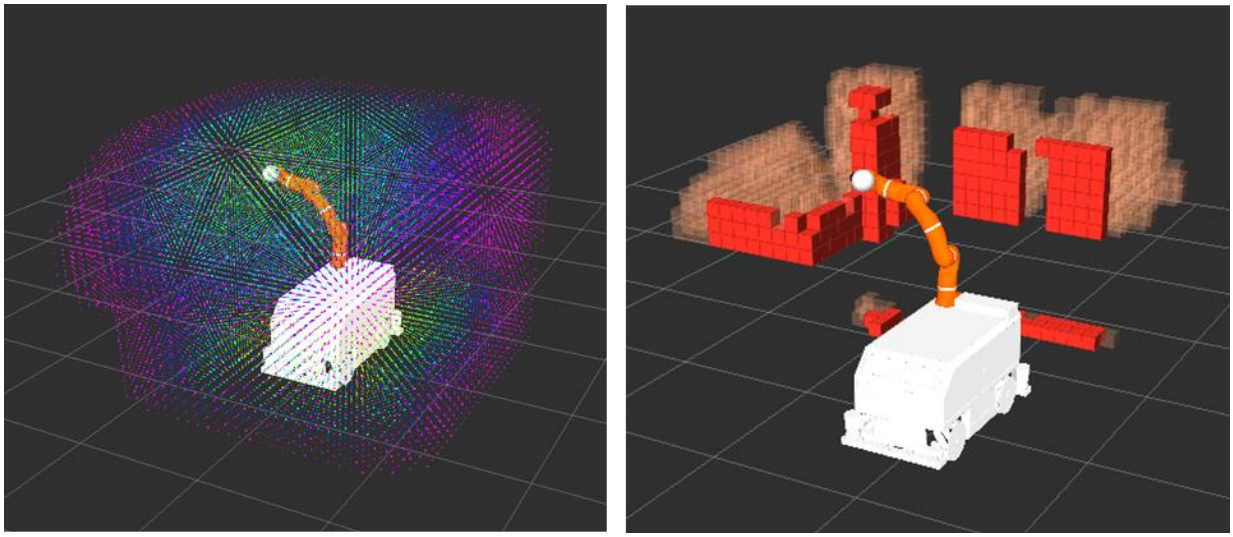


Figure 6 Reachability grid (left) and obstacle octree (right) with occupied (red) and occluded nodes (orange) of the simulated scene

In Figure 7 snap-shots containing the obstacle octree of real sensor data are shown. To the left of the robot, a human is walking and reaching out for the robot. On the other sides there are several static obstacles. For the reachability map and the obstacle octree the same parameters are used as in the simulation results. The computational times average 120ms for the reachability grid, 40ms for the URDF Filter, 35ms to compute the occupied and occluded obstacle nodes of one Kinect sensor, 7ms to compute the occupied and occluded obstacle nodes of one laser scanner, 25ms to compute the nodes occluded by the robot and 7ms to fuse the data in the resulting obstacle octree.

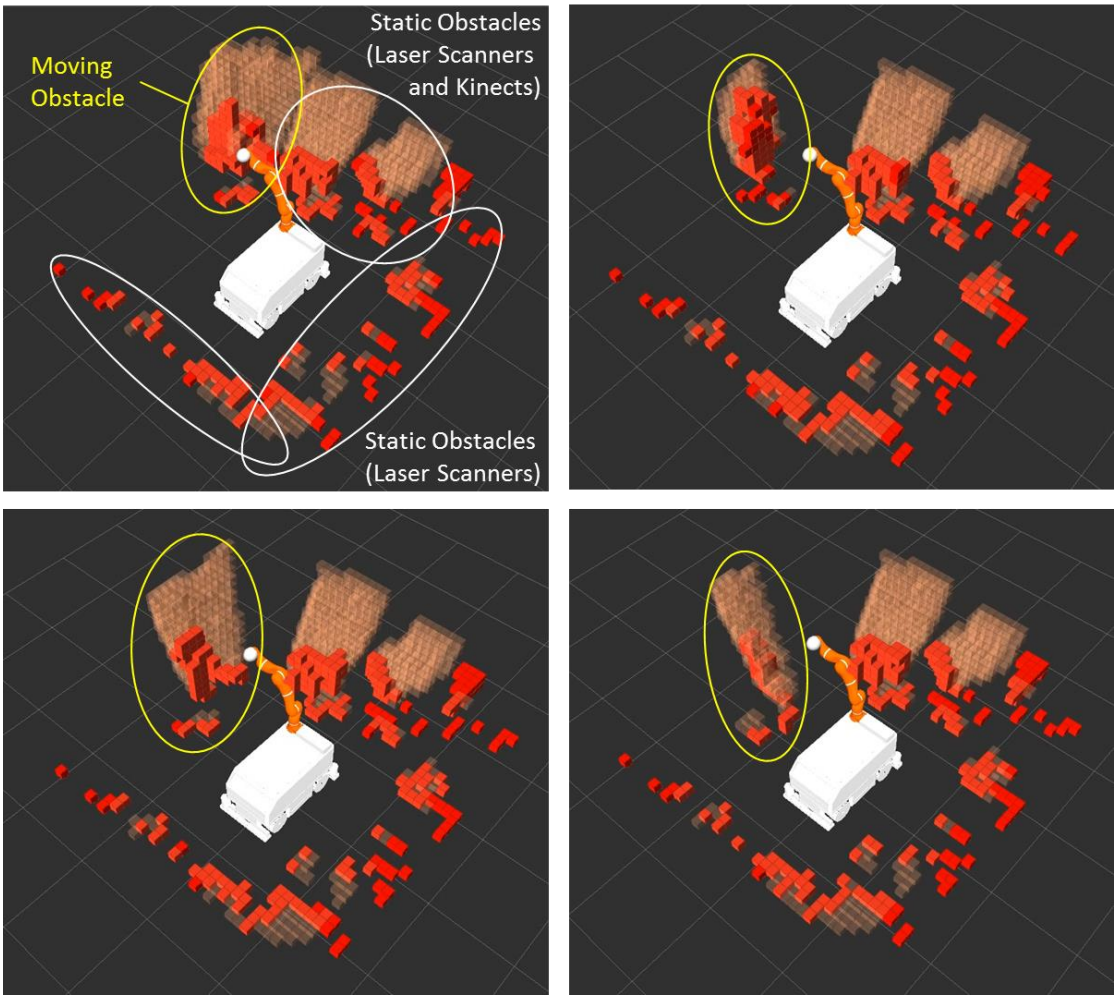


Figure 7 Snap-shots of experimental results (red: occupied nodes, orange: occluded nodes)

4 Collision detection through proximity/contact sensors

Safety issues become of primary concern when robot manipulators operate in an unstructured environment, where, unintended collisions should be avoided by anticipating dangerous situations, while the effects of actual collisions should be mitigated by a prompt reaction of the robot to recover a safe operative condition. Most of the solutions presented in literature, need for invasive actions or drastic changes in the existing robot architecture and in its software/electronic or mechanical parts, i.e., control algorithm, actuation system. Therefore, the use of "open" industrial manipulators that allows communicating directly with the low-level control architecture is advisable. However, most of the manipulators used today in industrial processes do not have this characteristic and it is difficult to intervene directly on the control unit. In the following, we will present a new approach based on a proximity/contact-force sensor integrated into a standard control unit of an industrial ABB robot.

4.1 The proximity/contact sensor

The sensor is able to detect both the presence of a nearby object and the contact pressure applied by an external object when a collision occurs. The sensor is interfaced with an ABB industrial robot using only the standard control unit; the standard RAPID primitives are used to define the robot task.

The proposed solution is based on optoelectronic sensing elements; each one consists of a couple of infrared Light Emitting Diode (LED) and Photo-Detector (PD). The elements can be used to implement both contact-force and proximity sensing elements. In particular, each contact-force sensing element was implemented by covering the selected taxel with a deformable silicone layer that transduces the external force into a mechanical deformation, measured by the photo-detector. For the implementation of the proximity sensing element, the same elements described above were used without the deformable layer. The light emitted by the LED is now reflected by the object surface; the light intensity received from the phototransistor and, thus the photocurrent, is inversely proportional to the squared distance between the two devices and the reflecting surface of the object, which allows getting a distance information.

Only two resistors constitute the complete conditioning electronics of the optoelectronic couple, one to fix the forward current of LEDs and the other to convert the measured photocurrent into a voltage signal. The taxels are organized in arrays of 7 elements for each module, which is equipped with a MPU. The latter (a PIC16F690 manufactured by Microchip) is equipped with an Analog-to-Digital (A/D) converter with a 10 bit resolution, in order to acquire the taxel measurements. The microcontroller has an SPI serial interface that supports the daisy-chain configuration that allows a high-speed data transmission along the module chain. The complete sensor is obtained by connecting a number of modules where the microcontroller of each module works as a slave device. The architecture is completed with a master module constituted by a microcontroller, which supports the same serial protocol of the slave modules.

The maximum distance at which an object can be detected by an optoelectronic proximity sensor also depends on the characteristics of its surface (i.e., color, roughness, and reflectivity). If the human skin represents the colliding object, the developed sensor is able to detect it in a range of about 5 cm. Surely, the nominal range of the proximity sensor should be adjusted with respect to the robot dimensions, working speed, robot location where the sensor is installed. In fact, a too wide range is not recommended because it could induce false-positive detections caused by self-collision conditions during the robot movements. With this methodology, a proximity/contact-force sensor can be developed alternating proximity and contact-force sensing elements.

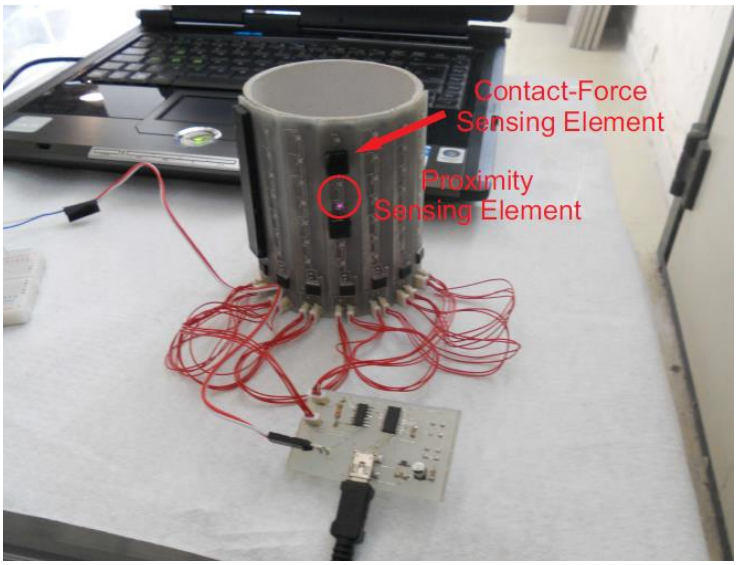


Figure 8 The proximity/contact sensor

Figure 8 reports a sensor prototype installed on a cylindrical support, where a proximity sensing element and a contact-force sensing element are highlighted. It is constituted by 7 modules that allow to obtain a complete sensor with 49 elements. For example, by implementing 5 proximity sensing elements (10 elements are necessary), the remaining 39 elements can be used to implement contact-force sensing elements, with a total of 44 mixed sensing points. More details about the sensor can be found in [5] and [6].

4.2 Interfacing with the ABB IRB 140 robot

The developed sensor has been installed on an ABB industrial manipulator in order to study and develop a new approach to improve safety in human-robot cooperative tasks that would be applicable to any industrial robot without introducing significant changes in the control unit. An ABB IRB 140 manipulator is used in the experiment. The IRB 140 uses an S4CPlus control unit. It contains the electronics required to control the manipulator, external axes and peripheral equipment. The controller also contains the system software, i.e. the BaseWare OS, which includes all basic functions for operation and programming. The robot complies fully with the health and safety standards specified in the EEC's Machinery Directives and it has a dedicated safety system based on a two-channel circuit, which is monitored continuously. The

S4CPlus controller does not authorize any access to the internal control logic. So, a Digital I/O 24VDC Unit connected to the distributed I/O system of the S4CPlus controller unit, based on the Fieldbus standard CAN/DeviceNet, has been used to communicate with external devices.

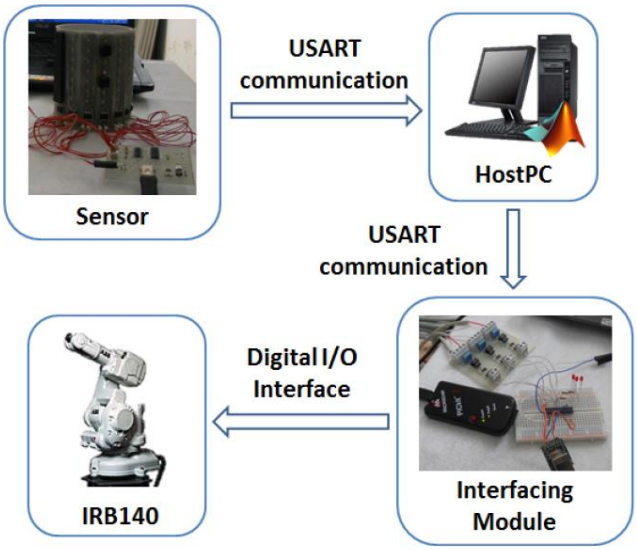


Figure 9 System architecture diagram

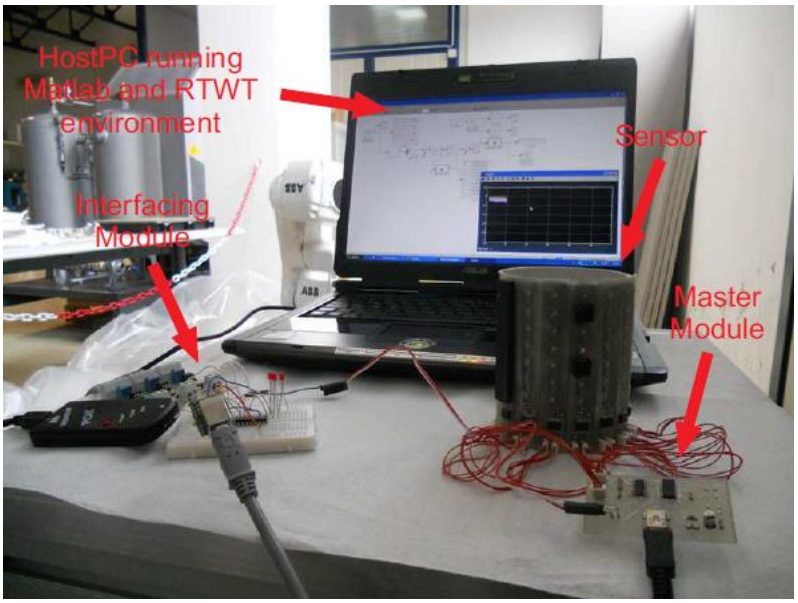


Figure 10 The experimental setup

Figure 9 shows a diagram of the system architecture used in the experiment, while in Figure 10 the experiment setup is reported.

4.3 The safety strategy for collision handling

The behaviour of the manipulator is as follows: the robot is programmed to follow a path at constant speed, the speed is reduced as soon as the proximity sensor detects an obstacle, the robot is stopped when

a contact occurs, the robot motion is resumed as soon as no contact occurs and the minimum safety distance is measured. The basic idea to implement such a behaviour is to create a mapping between the eight discrete level, obtained with the three input digital lines, and eight safety levels corresponding to different ranges of the proximity/contact-force sensor signals. The proximity/contact sensor communicates through the master module with a HostPC running an acquisition software executed in real time on Matlab and RTWT environment. The HostPC acquires the sensor data through the Simulink RTWT model, elaborates them and, on the basis of the data values, sends to an interfacing module the information on the safety level to adopt. The interfacing module consists of an elaboration unit, a microcontroller, that generates three digital output signals, which encode in a binary format the safety level received from the HostPC, and of a logic level converter board, which adapts the TTL voltage levels of the microcontroller to the PLC voltage levels of the robot digital interface. A *Gray code* was used instead of a *Natural binary code*, to encode the eight safety level, in order to avoid problems related to non-simultaneous switching of digital signals. On the basis of the sensor data and of the safety level provided by the HostPC the TCP linear speed is modified: the closer is the object, the slower is the speed. Seven of the eight levels (*level 0* to *level 6*) are assigned to seven different ranges of the proximity sensing element signal; while the *level 7* is activated when a contact with the object occurs. The safety strategy is summrize in Table 1.

Safety level	Gray code	TCP linear speed
0	000	400 mm/s
1	001	320 mm/s
2	011	260 mm/s
3	010	200 mm/s
4	110	140 mm/s
5	111	80 mm/s
6	101	10 mm/s
7	100	0 mm/s

Table 1 Safety levels - TCP linear speed associations

Some of the results of a typical experiment are shown in Figure 11, where the left plot reports the proximity sensor signal compared to the safety levels defined above, and the right plot reports the commanded TCP linear speed computed according to Table 1. Further details can be found in [6].

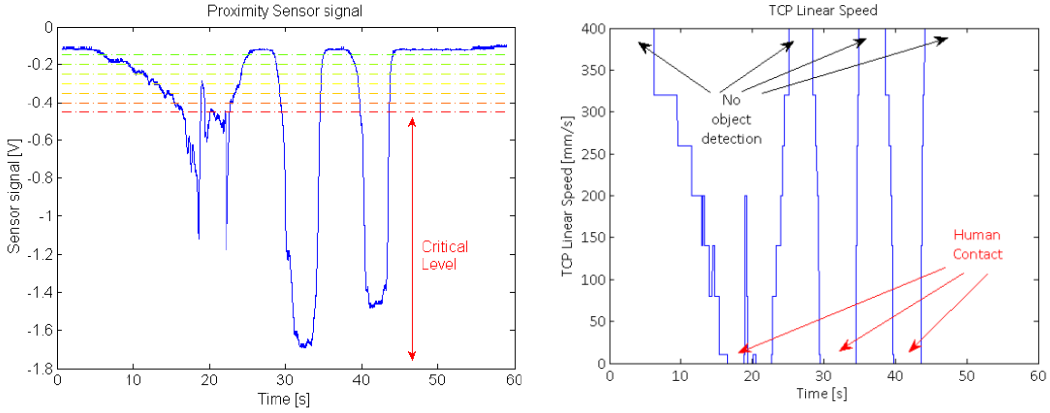


Figure 11 Proximity sensor signals (left) and commanded TCP speed (right)

5 The distributed proximity sensing approach

In the following, we will present a method to detect and avoid collisions on the basis only of simple proximity information coming from a number of sensors distributed on the robot without resorting to any sensorization of the environment. The method can be also seen as a technique able to improve flexibility of the motion planning process for mobile manipulators. In fact, proximity data are used to correct pre-planned motions to cope with uncertainties and dynamic changes of the scene at execution time. The algorithm computes robot motion commands aimed at fulfilling the mission by combining two tasks at the same time, i.e., following the planned end-effector path and avoiding collisions with obstacles in the environment, by exploiting robot redundancy as well as handling priorities among tasks. Moreover, a technique to smoothly switch between the tasks is presented. To show the effectiveness of the method, four experimental case studies have been carried out consisting in a place task executed by a youBot system [7] in an increasingly cluttered scene. The method has been proposed for the first time in [8], where all the details of the algorithm can be found. Only a brief summary is presented here.

5.1 The protective buffer

To detect collisions with objects the robot is considered as “protected” by N springs with a certain rest length, each one attached to a proximity sensor. When an obstacle “touches” one or more springs (in the sense that it comes close enough to the corresponding sensors), the total elastic energy associated to these springs increases. To accomplish the task without hitting any object, the robot path has to move away from the obstacle to minimize the total energy, i.e. the sum of the elastic energies of all the springs touched by the obstacle. In order to formalize the approach, assume that the k-th sensor, located in the point P_{s_k}, measures the distance d_k between the nearest point on the obstacle P_{o_k} and the sensor itself as well as the direction of the minimum distance expressed by the unit vector v_{d_k}. The method can be applied to a generic kinematic open chain, therefore the mobile base is assumed to be holonomic, as it is for the youBot used in the experiments presented in this work. Assuming to adopt the Denavit-Hartenberg (DH) convention and that all vectors expressed in frame 0 (the first fixed DH-frame of the chain) have no superscript, with reference to Figure 12, let

- $q = [q_1\ q_2 \dots q_n] \in \mathbb{R}^n$ be the vector of configuration variables;
- $e_4 = [0 \quad 0 \quad 0 \quad 1]^T$;
- $\tilde{p}_{i-1} = A_1^0(q_1)\cdots A_{i-1}^{i-2}(q_{i-1})e_4$ be the position vector (homogeneous coordinates) of the origin of the DH-frame fixed to the link *i*;
- $z_{i-1} = R_1^0(q_1)\cdots R_{i-1}^{i-2}(q_{i-1})z_0$ be the *z* axis (along joint *i*) of the frame fixed to link $i-1$;
- p_{o_k} be the position vector of the obstacle point at minimum distance from the k-th sensor;
- $p_{s_k}(q) = p_i(q) - R_i^0(q)dp_k^i$ be the position vector of the kth sensor point, being dp_k^i the (constant) position of the k-th sensor (assumed mounted on link i) with respect to frame i fixed to link i

- position of the k-th sensor (assumed mounted on link i) with respect to frame i fixed to link i

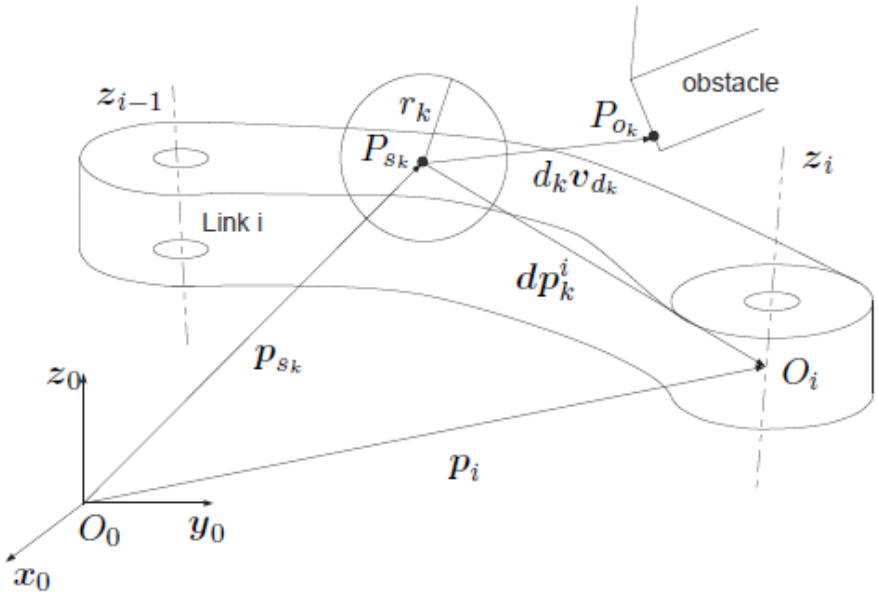


Figure 12 A generic link with a sensor point and an obstacle

Now, the pseudo-energy of the k-th spring is defined as

$$\varepsilon(\mathrm{q}) = \begin{cases} \frac{1}{2}(\mathrm{d}_k - r_k)^2, & \text{if } d_k \le r_k \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where $d_k(q) = \left\| p_{o_k} - p_{s_k}(q) \right\|$ is the information provided by the k-th sensor, together with the direction $v_{d_k}(q) = (po_k - p_{s_k}(q)) / \left\| p_{o_k} - p_{s_k}(q) \right\|$.

Note how the value of the pseudo-energy depends on both the distance between the obstacle and the robot and on the value r_k, defined by the user and interpreted as the rest length of the spring. The total energy of all the springs is simply:

$$\sigma(q) = \sum_{k-1}^{N} \varepsilon_k(q). \tag{5.2}$$

The objective of the algorithm is to follow the given reference trajectory $x_d(t)$ with the robot end effector avoiding collisions between any part of the robot with any other object in the environment, i.e. any other part of the robot and any obstacle fixed in unknown locations or subject to unknown motions.

5.2 The collision avoidance algorithm

To tackle the problem, a NSB approach is adopted with two tasks [9]. The first one consists in finding a trajectory in the configuration space so that the end-effector trajectory is followed, therefore the first task function is simply the direct kinematics function, i.e.

$$x = k(q), \tag{5.3}$$

where $x \in \mathbb{R}^r$ is the vector containing the task space variables. The second task consists in finding a trajectory in the configuration space so that the total pseudo-energy is zero, i.e. $\sigma_d = 0$.
According to the NSB approach, each of the two tasks above can be executed by actuating the corresponding velocity in the configuration space, i.e.

$$\dot{q}_g = J_g^{\dagger}(q)(\dot{x}_d + \gamma_g(x_d - k(q))) \tag{5.4}$$

$$\dot{q}_o = J_o^{\dagger}(q)(\dot{\sigma}_d + \gamma_o(\sigma_d - \sigma(q))) \tag{5.5}$$

where $J_g(q) = \partial k(q)/\partial q$ is the Jacobian of the robot, $J_o(q) = \partial \sigma(q)/\partial q$ is the gradient of the energy function and the symbol $X^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of the matrix X; γ_g and γ_o are the CLIK gains, which have an influence on the transient behavior of the robot and on the convergence of the CLIK algorithms. For a thorough discussion on the selection of these gains when the above algorithms are implemented in discrete time, the reader is referred to [10]. Note that for the task Jacobian J_g, instead of the standard Moore-Penrose pseudo-inverse, a weighted pseudo-inverse[1] can be computed so that motion of the arm degrees of freedom can be privileged with respect to base motion, that can help to save power consumption. The definition of the energy function in (5.1) and (5.2) allows the computation of the gradient J_o in closed form as

$$J_o(q) = \sum_{k-1}^{N} \frac{\partial \varepsilon_k(q)}{\partial q}, \tag{5.6}$$

where

$$\frac{\partial \varepsilon_k(q)}{\partial q} = \begin{cases} -(d_k - r_k) v_{d_k}^T \frac{\partial p_{s_k}(q)}{\partial q}, \text{if } d_k \le r_k \\ 0, \text{otherwise} \end{cases} \tag{5.7}$$

A key contribution of this work is the method to combine the two tasks. This is accomplished by the following convex combination

[1] $J_g^{\dagger} = W J_g^T (J_g W J_g^T)^{-1}$, with W usually selected as a positive definite diagonal matrix with higher entries in the places corresponding to the degrees of freedom with a desired higher speed.

$$\dot{q} = (1-\lambda(d))\,\dot{q}_g + \lambda(d)\left(\dot{q}_o + (I - J_o^{\dagger}(q)J_o(q))\,\dot{q}_g\right), \tag{5.8}$$

where d measures the distance between the closest obstacle and the sensor closest to this obstacle, i.e.

$$d = \min_{k=1,\ldots,N} d_k \tag{5.9}$$

and $\lambda(d)$ is a weighting function. In classical approaches, $\lambda(d)$ is chosen as a simple switching function, i.e.

$$\lambda(d) = \begin{cases} 1 & d \le f \\ 0 & d > f \end{cases} \tag{5.10}$$

where f is an activation threshold to be selected. It is well-known that this choice can generate a sort of chattering of the computed velocity (5.4). To avoid this undesired effect that can generate vibrations of the mechanical structure of the robot, $\lambda(d)$ can be chosen as a smooth sigmoidal function, i.e.,

$$\lambda(d) = \frac{1}{\pi}\arctan(-K(d-f)) + 1/2, \tag{5.11}$$

or a piece-wise linear function, i.e.,

$$\lambda(d) = \begin{cases} 1 & if \quad d \le f - \Delta/2 \\ 0 & if \quad d \ge f + \Delta/2 \\ 1/2 - (d-f)/\Delta & \text{otherwise} \end{cases} \tag{5.12}$$

Note that in this way a sort of ``gray zone'' is defined where apriority among the tasks is not ``crisply'' defined. Outside of this zone (namely for d large or small enough), the priority is established by the classical null-space projection method. The design parameters $K > 0$ and Δ vary the slope of the function and thus the width of this gray zone. Note that a similar method has been proposed to combine different tasks in [11] but there the tasks are combined in the tasks space while here the combination is implemented in the configuration space, which simplifies handling of priorities.

5.3 Experimental results

To test the proposed algorithm, a Kuka youBot mobile manipulator is considered. Figure 13 shows a picture of the youBot with the proximity sensors distributed all over the base and thearm. Ten proximity sensors have been mounted on the robot, in detail 6 Sharp GP2Y0A21YK have been mounted on the mobile base since they have a range of about 80 cm, while 4 SharpGP2D120XJ00F with 30 cm range have been mounted on the arm.

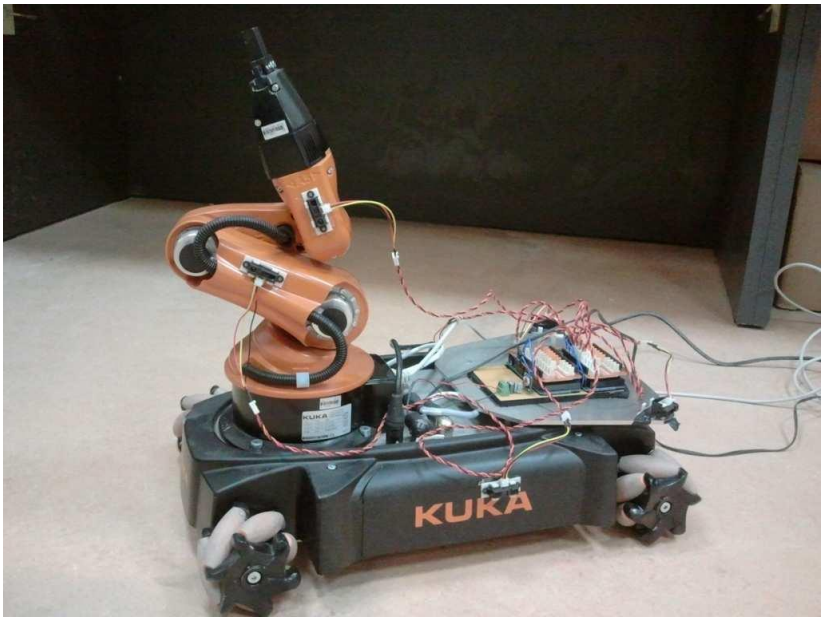


Figure 13 The Kuka Youbot with the proximity sensors installed.

The position of each sensor, expressed in the frame fixed to the link the sensor is attached to, is reported in the matrix

$$P = [\, p_{s1}^{l_1} \quad \dots \quad p_{s10}^{l_{10}} \,] = \begin{pmatrix} 16 & 23 & 10 & -4 & -7 & 0 & 4 & -29 & -29 & 0 \\ -11 & 0 & -16 & -16 & -2 & 6 & 0 & -13 & 13 & 0 \\ 0 & 0 & 4 & 0 & 8 & 4 & 7 & 5 & 5 & -8 \end{pmatrix} \tag{5.13}$$

where the i-th entry li of the vector $l = [l_1 \dots l_{10}]$ is the index of the link on which the i-th sensor is mounted, and the 10 sensors are mounted such that $l = [3,3,3,3,5,7,7,3,3,6]^T$. Similarly, the unit vector of each sensor optical axis expressed in the same frame is reported in the matrix

$$V = [\, v_{d1}^{l_1} \quad \dots \quad v_{d10}^{l_{10}} \,] = \begin{pmatrix} \frac{\sqrt{2}}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 1 & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{3}}{2} & -1 & 0 & 0.98 & 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.2 & 0 & 0 & 0 & -1 \end{pmatrix} \tag{5.14}$$

Note that the positions and directions of the sensors have been selected to cover the widest volume around the robot. Moreover, the sensors on the arm were located such that no sensor detects any robot link as an obstacle in the initial configuration selected to carry out the tasks described below. The sensors have been interfaced with the robot control PC through a serial communication channel by connecting the

sensor analog outputs to the A/D channels of two microcontroller boards ARDUINO Mega2560. On each microcontroller runs a ROS node that publishes messages containing sensor data at a rate of approximately 300Hz.

Figure 14 Comparison between arm configuration with and without the box in the first case study.

In order to show the performance of the developed algorithm, a task and four case studies have been selected. The task consists of placing an object into a box and it is composed by the following steps:

- the robot opens the gripper
- a human gives an object to the robot
- the robot closes the gripper
- the robot brings the object into the box

The first case study consists of the execution of the task without any obstacles, i.e., with the workspace completely empty. It is important to stress, in fact, that also the box in which the robot has to put the object can be an obstacle itself. Figure 16 shows the velocity commands which the flexible planner sends to the robot base and to the robot arm. In this case, it is evident that the commanded trajectories are simply the velocities in the configuration space such that the end effector follows the off-line planned trajectories in the Cartesian space. The task has then been executed in presence of the box, the flexible planner computes a correction of the off-line planned trajectory that brings the arm in a more stretched configuration (see Figure 14).

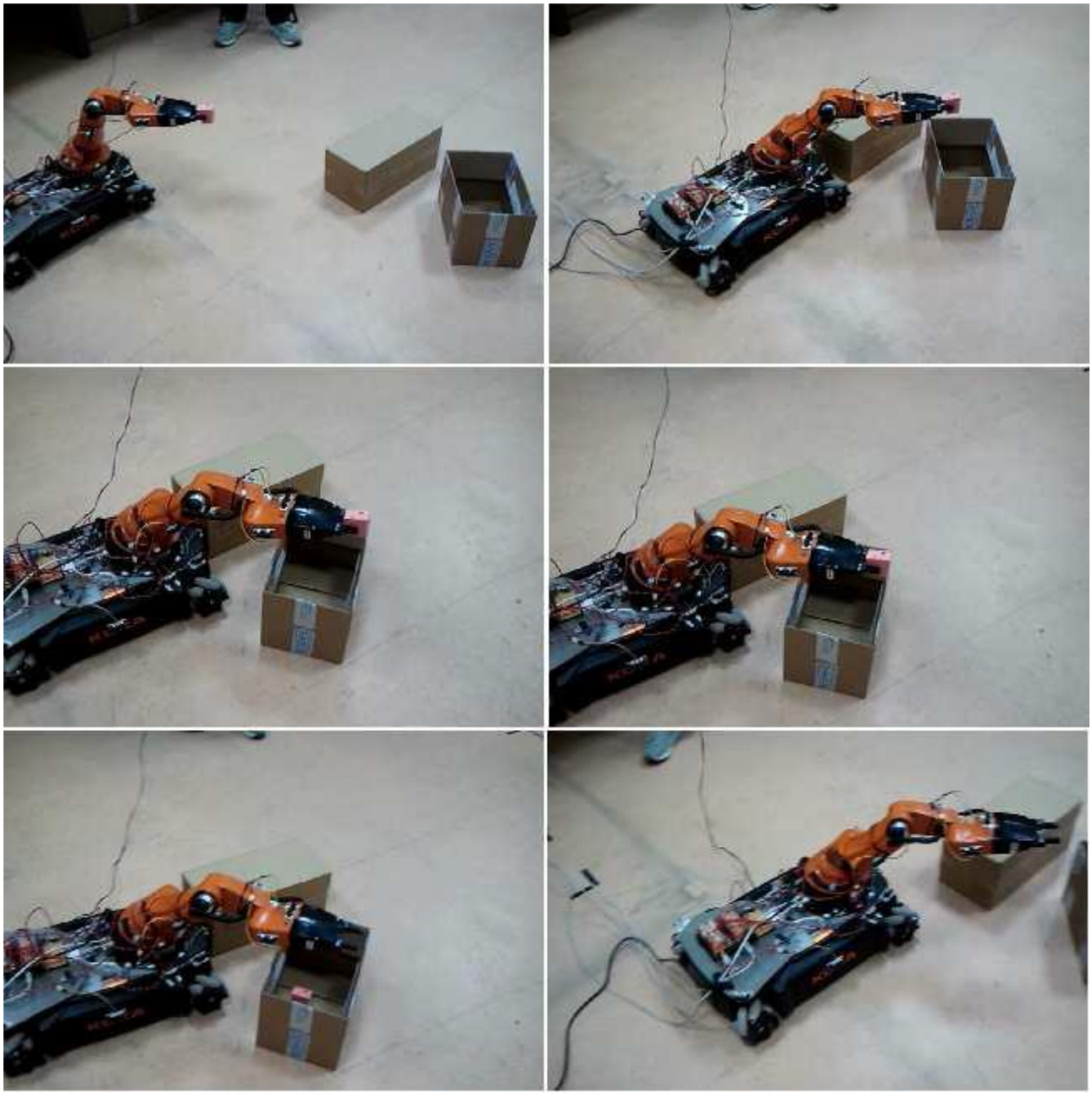


Figure 15 Snapshot sequence relative to the place task without enabling the flexible planning algorithm

This allows the base to stay at a safe distance from the box and it allows the end effector to bring the object in the target position at the same time. In the second case study (Figure 15 - Figure 17), an unexpected obstacle is placed between the box and the robot, but the flexible planning module is not enabled, that is the robot has to execute it without any modification. In more detail, Figure 16 reports the outputs of the most relevant sensors and the pseudo-energy is plotted. It is interesting to note that in the range 30 − 60 s, the normalized outputs of two sensors get close to 1 (the maximum value) and the pseudo-energy dramatically increases. This indicates that the robot has collided with the obstacle, as it is evident from the intermediate snapshots of Figure 15.

In the third case study, shown in Figure 17, the obstacle is still present but the flexible planning algorithm is enabled. Note how in Figure 18 both the pseudo-energy and the sensor outputs assume significantly lower values than in the previous scenario (that means the obstacle is at a greater distance). This is due to the trajectory corrections applied through the velocity commands. From the snapshot sequence reported in Figure 17, it is evident that no collision occurs. The last snapshot shows the robot configuration at the end of the task, which is with the end effector in the same position as at the task start. Interestingly enough, this configuration is totally different from the initial one, as the redundancy resolution method base on the jacobian pseudo-inverse is not cyclic. To show how the regularized switching function used to exchange priority among tasks is useful to avoid chattering of the velocity commands, the same task has been carried out using the switching function in (5.12).

As a last case study, the task has been executed with the robot acting in a more cluttered scene, which contains complex obstacles that obstruct the planned path not only from lateral directions but also from above. Figure 20 shows the usual snapshot sequence of the task execution. Note how the algorithm is able to compute corrections to the planned motion in every direction so that no collision occurs. In particular, owing to the proximity sensors on the arm, it gets down to keep a safe distance from the horizontal beam. In Figure 19 it can be be seen that there are more significant sensors than in the other cases. This is due to the presence of more obstacles in the workspace.

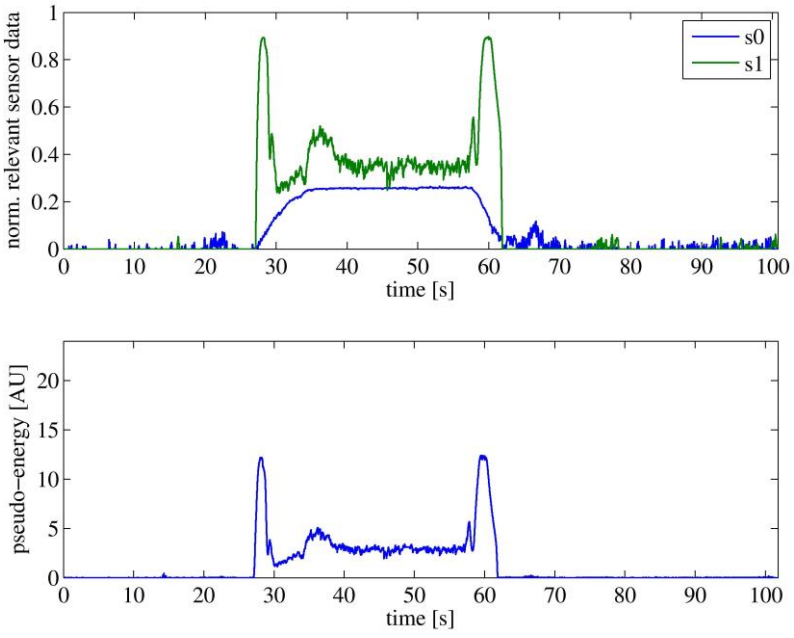


Figure 16 Normalized output of the relevant proximity sensors (top) and pseudo-energy (bottom) in presence of an obstacle and without enabling the flexible planning algorithm

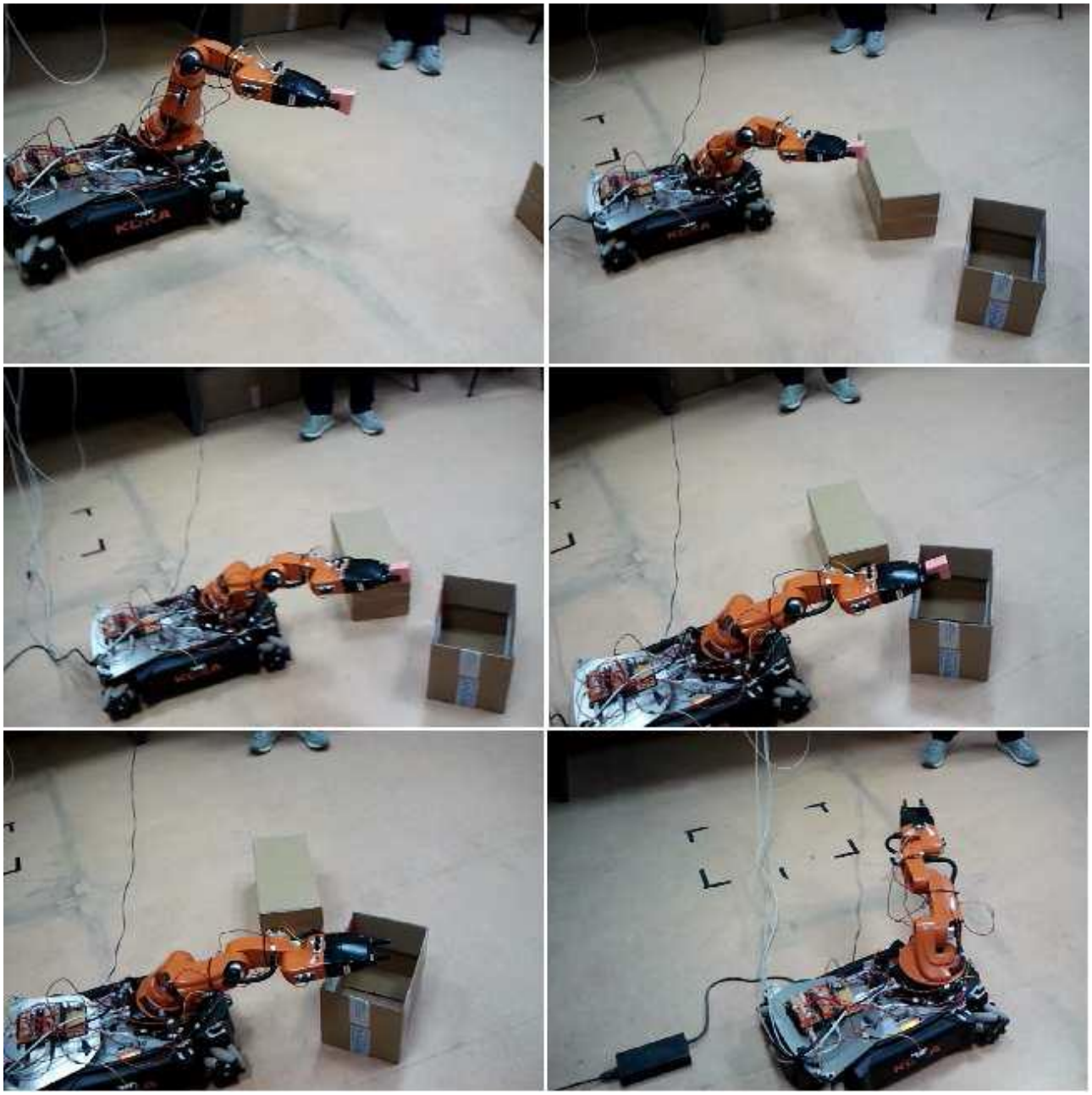


Figure 17 Snapshot sequence relative to the place task in presence of an obstacle and with the flexible planning algorithm enabled

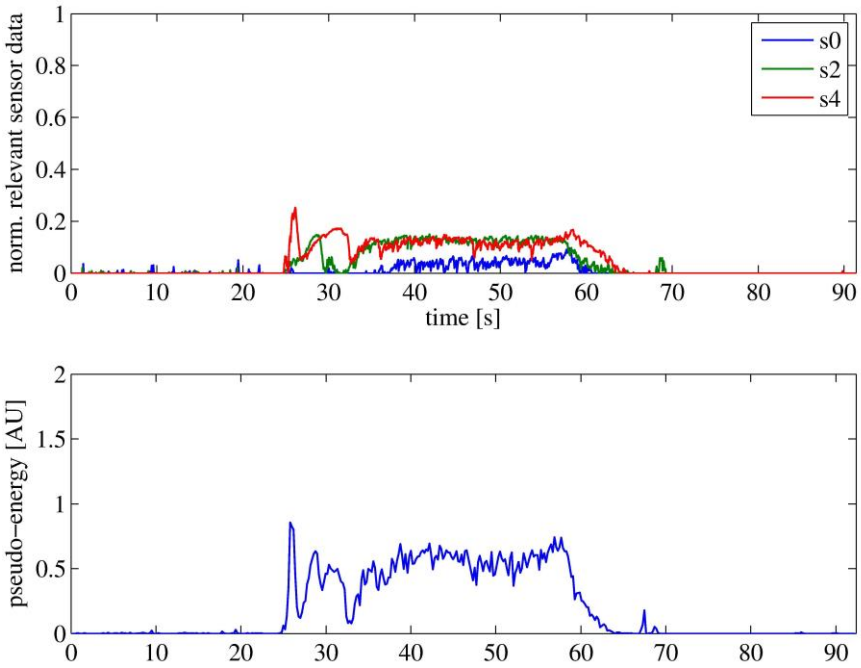


Figure 18 Normalized output of the relevant proximity sensors (top) and pseudo-energy (bottom) during the place task with obstacle

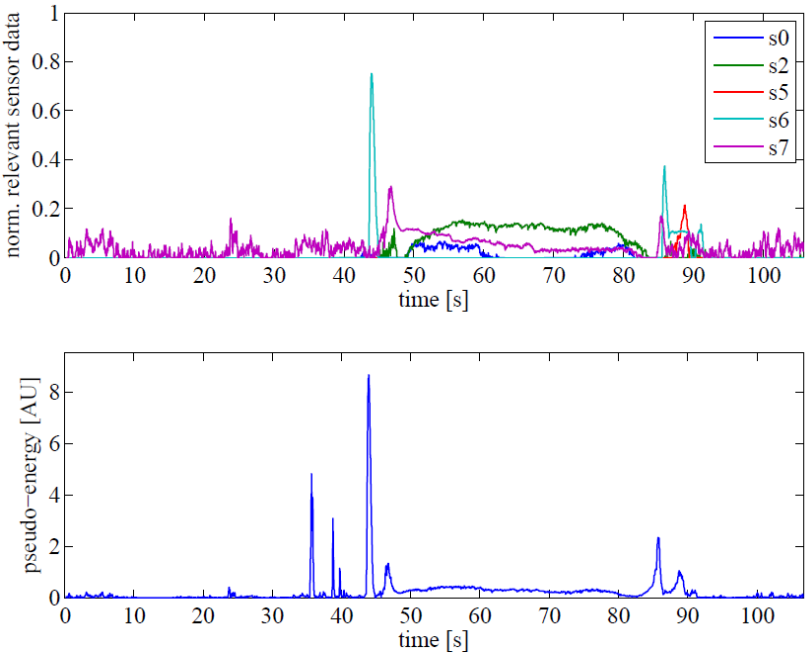


Figure 19 Normalized output of the relevant proximity sensors (top) and pseudo-energy (bottom) during the place task executed in the cluttered environment

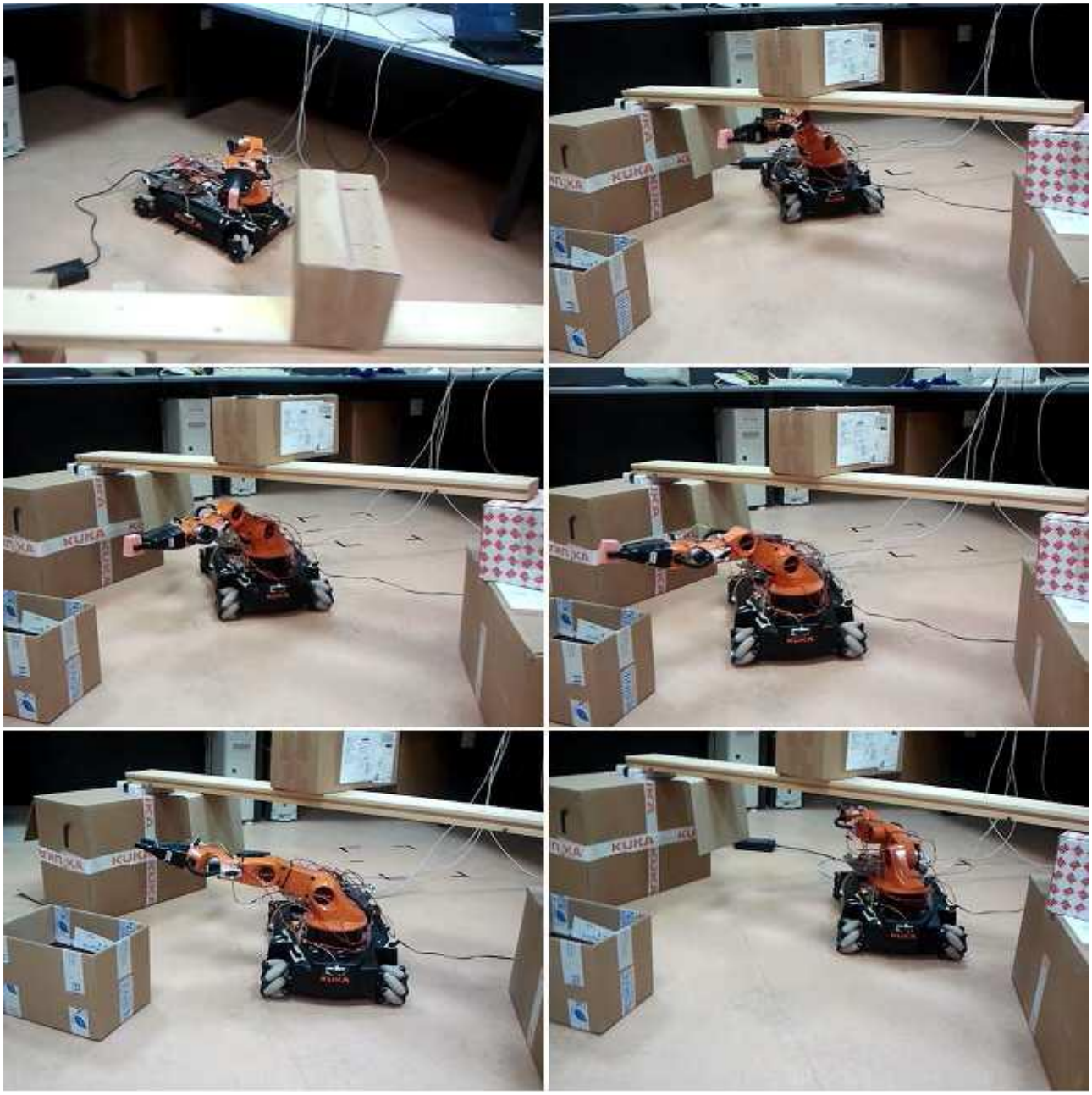


Figure 20 Snapshot sequence relative to the place task in a cluttered scene

The results here reported demonstrated that simple proximity information can be actually exploited for setting up a method for collision detection and avoidance even in cluttered environments. Commercial proximity sensors have been successfully exploited to modify the planned trajectories of a mobile manipulator with 8 degrees of freedom and adapt them to the actual scene in which the robot acts. This means that it is not necessary that the planner requires an accurate model of the scene, since uncertainties on the location of obstacles, that can even be moving, can be fully handled at the control level. Further advantages of the approach rely on the low computational burden of the flexible planning algorithm and on the exploitation of only very simple sensory information, which means that higher sampling rates are

possible thus improving dynamic performance. The limitation of the method is basically its local validity, i.e. the robot is not guaranteed to fulfil the complete task in any scene configuration.

6 Contact force estimation and contact point detection

When the human operator collaborates with a redundant robot, the redundancy of the robot, opportunely controlled, permits to obtain robot capabilities essentials for safe physical human-robot collaboration. We concentrate on the physical human-robot collaboration phase, in which a human enters in contact with a robot, and the robot reacts as a function of the exchanged forces. For this to happen, the robot has to accomplish four subtasks, which require fusing proprioceptive and exteroceptive sensing information:

1) detect a contact with the human, and distinguish between intentional contact or undesired collision;
2) identify the point on the robot surface where the contact has occurred;
3) estimate the exchanged Cartesian forces;
4) control the robot to react according to a desired behaviour.

Combining the geometric information about the contact area, as localized online by a depth sensor [12], with the joint torques due to contact, as obtained by our standard residual-based method [13], allows estimating the exchanged Cartesian forces at any contact point along the robot structure, without the need of force or torque sensing. The estimate of the contact force can be used in the design of admittance, impedance, or force control laws. In all cases, the reference behaviour can be conveniently specified and controlled at the actual contact, rather than at the joint or end-effector levels.

6.1 Contact forces estimation

Physical collaboration is characterized by force exchanges between human and robot, which may occur at any place of the robot structure. Detection of these contacts is a fundamental feature for safe pHRI and must be very efficient, in order to allow a fast robot reaction. We use a scheme based on residuals that detects physical contact and provides the joint torques associated to the external force [13]. This method needs an accurate dynamic model, but uses only robot joint position provided by the encoders.

Based on the generalized momentum of the robot $\boldsymbol{p} = \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$, where $\boldsymbol{M}(\boldsymbol{q})$ is the symmetric, positive definite inertia matrix, and $\boldsymbol{q}$ is the generalized coordinate vector of the robot.

We define the residual signal

$$\boldsymbol{r}(t) = \boldsymbol{K}_I \left(\boldsymbol{p}(t) - \int_0^t \left(\boldsymbol{\tau} + \boldsymbol{C}^T(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{r} \right) ds \right)$$

where $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the Coriolis and centrifugal terms, $\boldsymbol{g}(\boldsymbol{q})$ is the gravity vector, and $\boldsymbol{\tau}$ the motor torque. Considering the standard dynamic of a robot it is easy to check that

$$\dot{\mathbf{r}} = \mathbf{K}_I (\boldsymbol{\tau}_{ext} - \mathbf{r})$$

Each component of $\mathbf{r}$ is an decoupled, first-order, unity-gain filtered version of the unknown external torque, thus can be used to detect a collision.

The estimation phase is activated when a soft contact is detected [14], that is $\|r\| > r_{th}$ being $r_{th} > 0$ a suitable scalar threshold used to prevent false detection due to measurement noise and/or model uncertainties on r. This signal, by suitably filtering, is obtained from the FastResearchInterface of the KUKA controller.

Suppose a contact along the robot structure with a generalized force $F_c \in \Re^3$ applied by the human to the robot. The residual vector contains an indirect estimate of F_c, being

$$r \approx \tau_{ext} = J_c^T(q) F_c$$

Therefore, the external force can be approximately estimated by pseudoinversion as

$$\hat{F}_c = \left(J_c^T(q)\right)^{\#} r$$

Note that the estimate will be limited to only those component of F_c that can be detected by the residual r. All forces $F_c \in \aleph(J_c^T(q))$ will not be recovered in $\hat{F}_c$. However, this should not be considered as a serious limitation since such force components do not produce active work, and are absorbed by the robot structure.

More in general, one can consider multiple simultaneous contacts, e.g. two contacts with human hands. In this case,

$$\begin{pmatrix} \hat{F}_1 \\ \hat{F}_2 \end{pmatrix} = \left(J_1^T(q) \quad J_2^T(q)\right)^{\#} r$$

6.2 Contact point detection

In the case of physical human-robot collaboration, intentional contacts occur usually between the human hands and the robot. We focused therefore on this case. However, it is easy to extend the approach to the consideration of the whole human body.

The contact point p_c is in general unknown. Most of the times it is assumed that the robot end-effector is the only possible contact point. This is clearly a limitation for human-robot interaction, constraining the possible applications.

Different ways exist to find the contact, e.g., by using a tactile sensor skin distributed along the whole surface of the robot. However, our goal is to identify the contact point in the less invasive way.

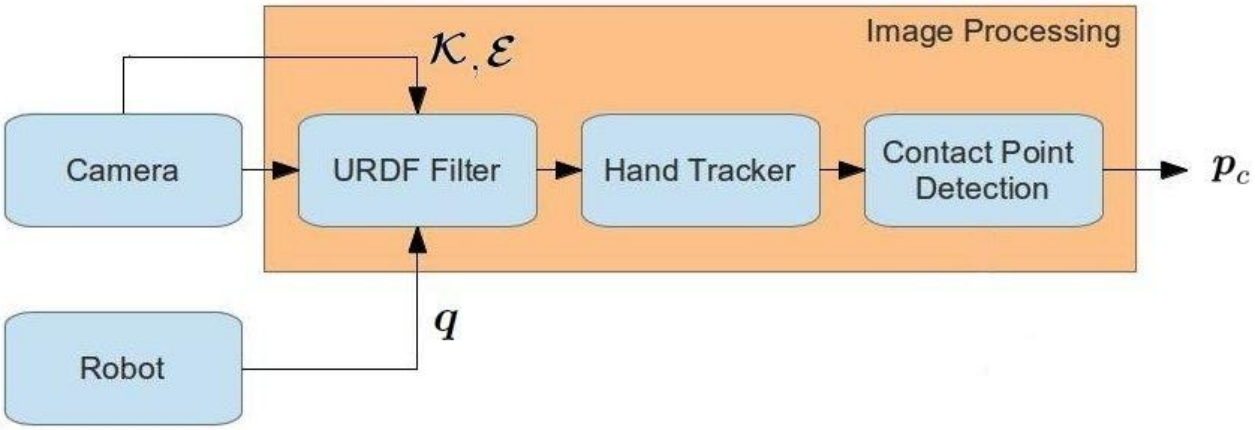


Figure 21 Image processing scheme

In our method (Figure 21), we find the contact point using a depth filtered image of the environment captured by a Kinect sensor, which is modelled as a classic pin-hole camera. The pin-hole camera model is composed by two sets of parameters, the intrinsic parameters, which model the projection of a Cartesian point on the image plane, and the extrinsic parameters, which represent the coordinate transformation between the reference frame and the sensor frame.

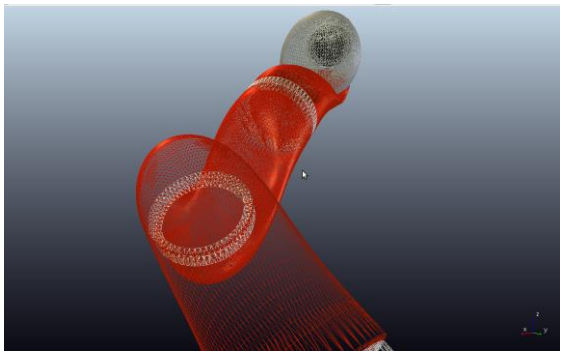


Figure 22 KUKA LWR structure modeled using a CAD software. Triangles are used as primitive shapes

A filter is used to remove the robot from the image starting from its URDF model. Each link of the robot is modelled as a set of primitive shapes having a certain number of vertices. In Figure 22 is shown how the KUKA LWR IV was modelled using triangles as primitive shape. The result is a discretisation of the real surface of the robot. The filtering operation is done by rendering the URDF robot model in the 3D depth scene, based on the joint position provided by the robot. The rendering capability of the graphic card is then exploited. This step is needed to avoid ambiguities between the robot and the human hand when they get very close.

The algorithm tracks the human hands and provides the associated vector position p_h. Thus, the distance between the hand and all vertices is computed. When a contact is detected, the vertex at minimum distance will be identified as contact point. Note that the contact point is also updated when the hand slides along the structure of the robot. The algorithm is presented below in pseudo-code form.

Algorithm 1 Contact point detection

```
min_distance = ∞;
contact_link = 0;
p_h=hand_position;
p_c=0;
while i < number_of_link do
  k = 0;
  while k < number_of_vertex do
    distance = ||p_h − mesh[i].vertex[k]||;
    if distance < min_distance then
      p_c = mesh[i].vertex[k];
      min_distance = distance;
      contact_link = i;
    end if
  end while
  i++;
end while
```

6.3 Experiments

Experiments have been performed on the KUKA LWR-IV manipulator having n=7 revolute joints, under a control cycle of 5 ms [15]. The workspace is monitored by a Microsoft Kinect depth sensor, positioned at a distance of 2.8 m behind the robot and at a height of 1.6 m with respect to the robot base frame. The Kinect captures a 640 x 480 depth image with a frequency of 30 Hz. The algorithm is executed on a quad-core CPU.

The robot is controlled at the joint velocity level with a classical admittance control scheme. In Experiment 1 the parameters of the controller are chosen for assigning a soft reactive behaviour to the robot. Instead, in Experiment 2 the parameters are chosen so as to obtain a rigid behaviour of the robot.

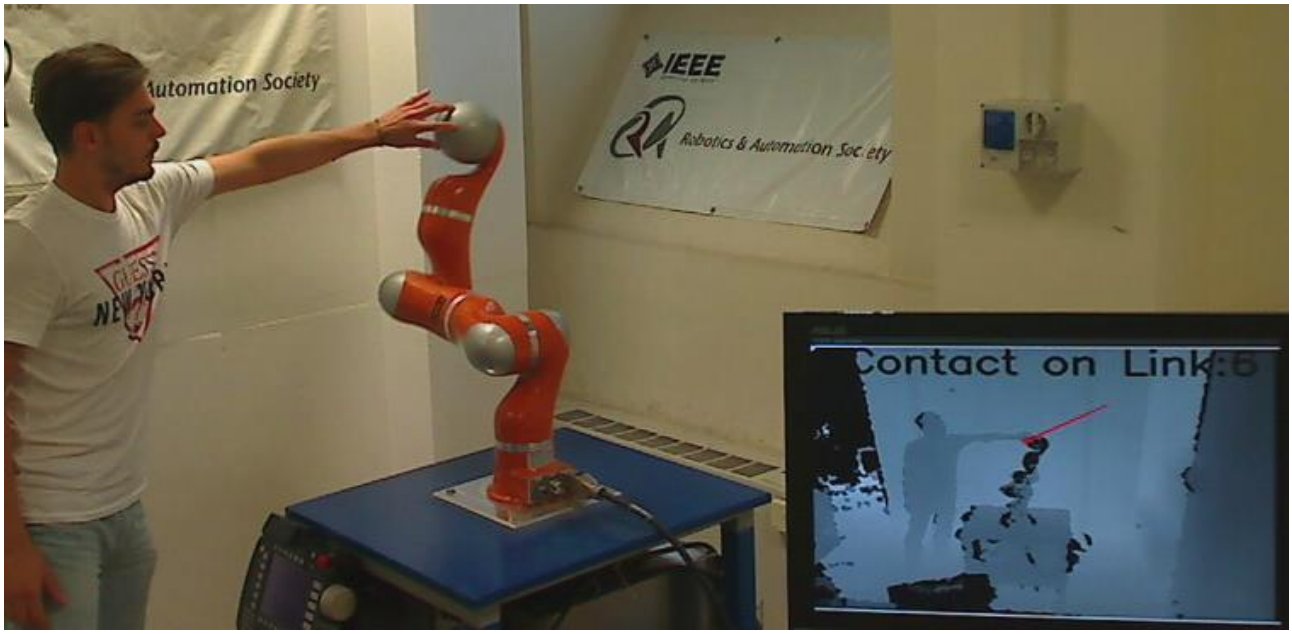


Figure 23 **Contact between human and robot on link 6 during a soft collaboration. Depth image with the estimated contact force vector highlighted in red (bottom right).**

In Experiment 1, the human pushes the robot on link 6 as shown in Figure 23. Figure 24 shows the components of the residual vector provided by the KUKA FRI, the associated estimated contact force, and the Cartesian position error of the contact point with respect to its initial position (when contact was established at t=0).

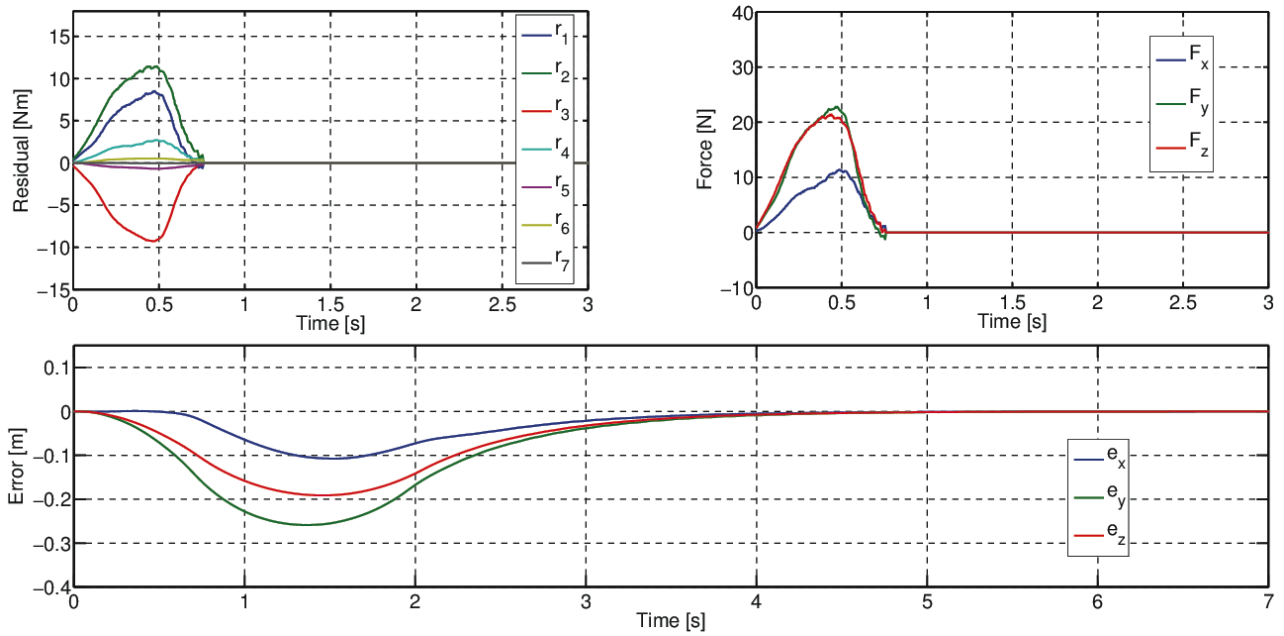


Figure 24 **Experiment 1: Human contact on link 6 starts at t = 0 and ends at t = 0.74. Residual vector components (top left), estimated force components (top right), and Cartesian position error components (bottom)**

After the detected contact, the robot moves the contact point along the direction of the estimated force. When the contact is no longer present, the contact point returns to its initial position. Note that, due to the imposed soft behaviour of the robot, the position error does still increase for a while even after contact has ended (t=0.76).

Figure 25 Balanced force on link 3 along the z-axis between human and robot during a collaboration with stiffer parameters. Depth image, with the vector of estimated contact force highlighted in red (bottom right).

In Experiment 2, the human pushes the robot on link 3 by applying a force along the z-axis (Figure 25). Contact is maintained for about five seconds, and during this interval the human force and the robot reaction force balance each other. The control parameters are chosen to obtain a stiffer behaviour. Figure 26 shows the residual vector components. Note that the second joint is more stressed than the others. Figure 27 and Figure 28 confirm the existence of a high force along the z-axis. In fact, both the force F_z and the error e_z are larger than the other components. Furthermore, the rigid behaviour of the robot causes a faster error transient than in the soft case.

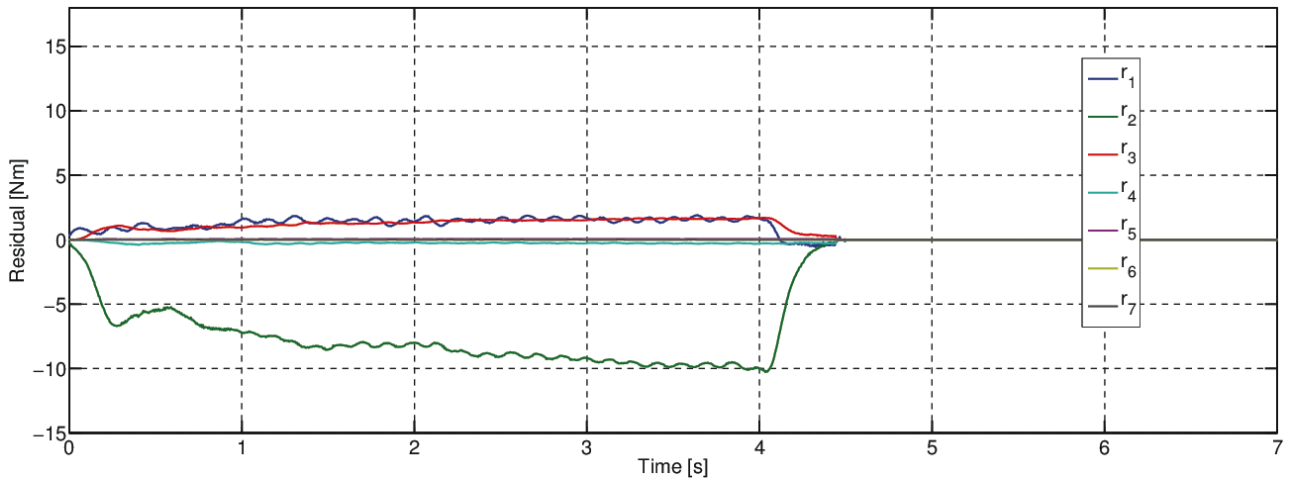


Figure 26 Experiment 2: Residual vector components. The contact on link 3 starts at t = 0 and ends at t = 4.65.

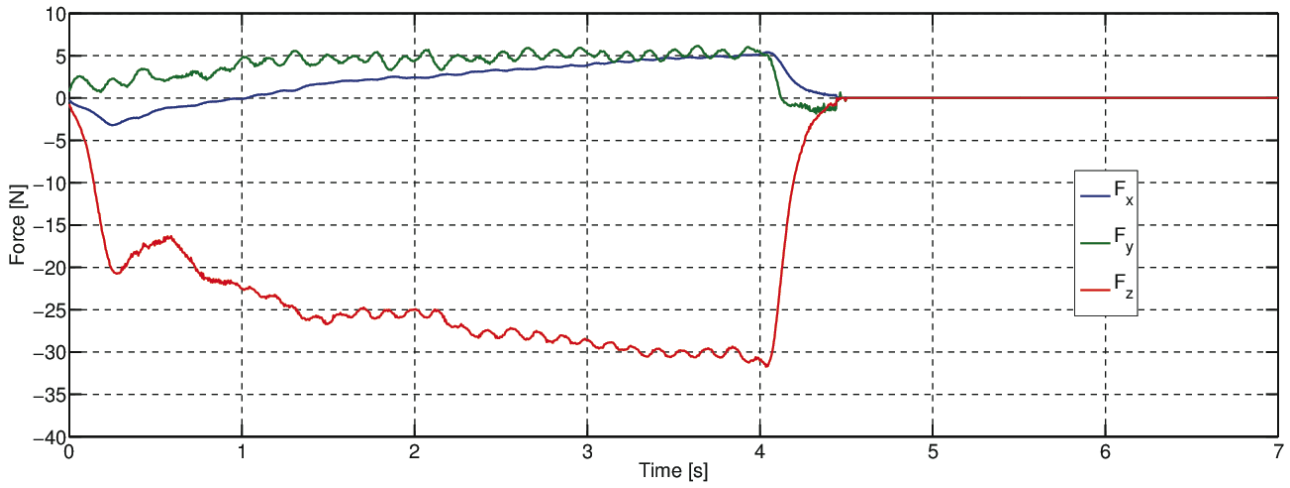


Figure 27 Experiment 2: Estimated Cartesian force components

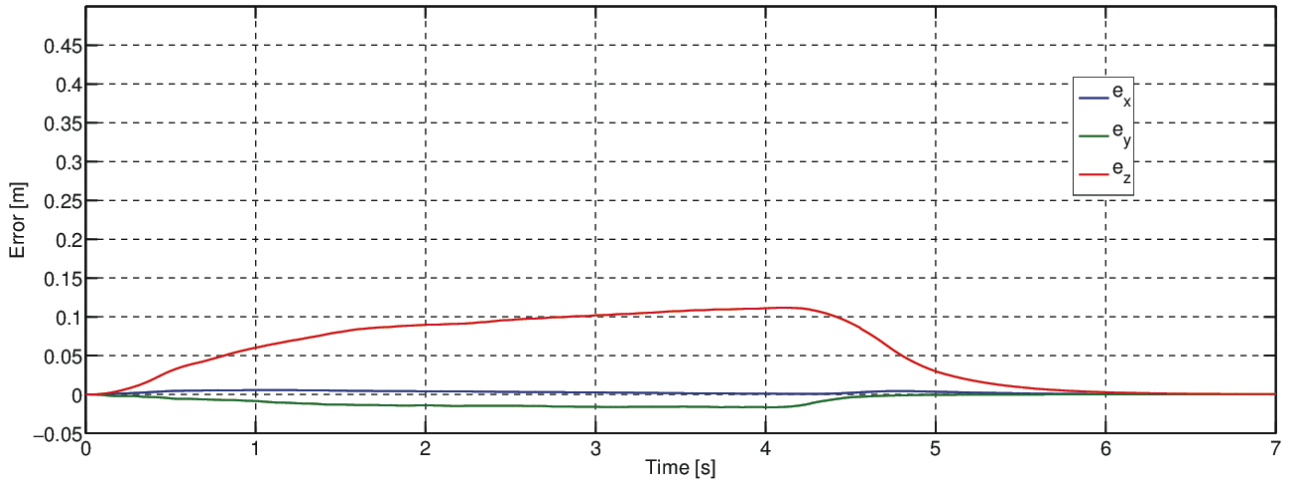


Figure 28 Experiment 2: Cartesian position error components.

7 Conclusions

In this deliverable, we have presented methods for close range monitoring during human-robot interaction based on different sensors. The methods cope with situations where humans and other obstacles are in the robot's close range and unintended collisions have to be avoided as well as with situations in which direct physical contact is intended.

Obstacles and their occlusions are represented in an octree structure to determine the minimum distance between the robot and obstacles. Distributed proximity sensors are used to build a protective buffer around the robot for collision avoidance. Contact forces and their contact points are estimated based on the measured joint positions and a Kinect sensor.

The developed close range monitoring algorithms identify sudden dangers within the workspace of the robot. They have been successfully tested on experimental robot and sensor setups.

8 References

[1] P. Anderson-Sprecher and R. Simmons, “Voxel-based motion bounding and workspace estimation for robotic manipulators,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, St. Paul, USA, 2012.

[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots,* 2013.

[3] I. A. Sucan and S. Chitta, "MoveIt!," Oct. 2013. [Online]. Available: http://moveit.ros.org.

[4] "Realtime URDF Filter," [Online]. Available: https://github.com/blodow/realtime_urdf_filter. [Accessed Oct. 2013].

[5] A. Cirillo, P. Cirillo and S. Pirozzi, “A Modular and Low-Cost Artificial Skin for Robotic Applications,” in *The Fourth IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*, Roma, 2012.

[6] A. Cirillo, P. Cirillo, G. D. Maria, C. Natale and S. Pirozzi, "A Proximity/Contact-Force Sensor for Human Safety in Industrial Robot Environment," in *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Wollongong, AU, 2013.

[7] R. Bischoff, U. Huggenberger and E. Prassler, "KUKA youBot - a mobile manipulator for research and education," in *IEEE International Conference on Robotics and Automation*, 2011.

[8] P. Falco and C. Natale, "Low-level flexible planning for mobile manipulators: a distributed approach," *Advanced Robotics,* 2013 (submitted for publication).

[9] G. Antonelli, F. Arrichiello and S. Chiaverini, "The null-space-based behavioral control for autonomous systems," *Intelligent Service Robotics,* 2007.

[10] P. Falco and C. Natale, "On the Stability of Closed-Loop Inverse Kinematics Algorithms for Redundant Robots," *IEEE Transactions on Robotics,* 2011.

[11] J. Lee, N. Mansard and J. Park, "Intermediate desired value approach for task transition of robots in kinematic control," *IEEE Transaction on Robotics,* 2012.

[12] F. Flacco, T. Kröger, A. De Luca and O. Khatib, “A depth space approach to human-robot collision avoidance,” in *2012 IEEE Int. Conf. on Robotics and Automation*, 2012.

[13] A. De Luca, A. Albu-Schäffer, S. Haddadin and G. Hirzinger, “Collision detection and safe reaction with the DLR-III lightweight robot arm,” in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[14] M. Geravand, F. Flacco and A. De Luca, “Human-robot physical interaction and collaboration using an industrial robot with a closed control architecture,” in *2013 IEEE Int. Conf. on Robotics and Automation*, 2013.

[15] E. Magrini, F. Flacco and A. De Luca, “Estimation of contact force using a virtual force sensor,” in *submitted to 2014 IEEE Int. Conf. on Robotics and Automation*, Hong Kong, June 2014.