Open Architecture for Vision-Based Robot Motion Planning and Control

Theodor Borangiu, Florin Anton and Silvia Anton

Abstract This chapter introduces a methodology for the vision-based motion control of robot manipulators. The motion control problem is decomposed into three computational stages: motion planning, trajectory generation and trajectory tracking. While the two latter activities are always executed in real time, motion is planned in traditional robot systems off line, by learning robot points or by using numerical output data from programs that plan minimal paths, avoid obstacles, etc. Guidance vision is introduced as an advanced motion control method, which provides flexibility when integrating industrial robots in computer-controlled manufacturing structures. A dynamic look-and-move system architecture is discussed, as a robot-vision system which is closed at task level. An open architecture is proposed as implementing solution for vision-based scene management and robot guidance, which integrates any types of robot controllers and image processing libraries. The chapter also presents a motion control algorithm for robots which are required to pick objects randomly moving on conveyor belts. The algorithm for visual tracking of conveyor belts for "on–the-fly" object grasping is partitioned in two stages: (i) visual planning of the instantaneous destination of the robot, (ii) dynamic re-planning of the robot's destination while tracking the object moving on the conveyor belt. The ensemble [conveyor belt + actuator + sensor] is configured as a single-axis Cartesian robot, leading to a cooperation problem between robot manipulators subject to multitasking control. Experimental results are finally reported in what concerns the statistics of object locating errors and motion planning errors function of the size of the objects of the belt speed and of the light strobe.

Keywords Robot-vision system · Vision guided robot planning · Visual robot servoing · Joint-space trajectory planning

T. Borangiu (✉) · F. Anton · S. Anton
Department of Automation and Applied Informatics,
University Politehnica of Bucharest, Bucharest, Romania
e-mail: theodor.borangiu@cimr.pub.ro

G. Carbone and F. Gomez-Bravo (eds.), *Motion and Operation Planning of Robotic Systems*, Mechanisms and Machine Science 29,
DOI 10.1007/978-3-319-14705-5_3

1 Introduction

The motion control problem refers to controlling the robot manipulator such that it follows a pre-planned path. The motion control problem is generally decomposed into three computational stages (Fig. 1): (1) Motion planning; (2) Trajectory generation; (3) Trajectory tracking [1].

In the motion planning stage, desired paths are described in the r-dimensional task space T (i.e. the locus of the positions and orientations that the robot tool must attain in $O \subseteq \mathsf{R}^m$—the operational space, $\mathcal{T} \subseteq O$), which is isomorphic to the special Euclidian group $\mathbf{SE}^3$.

$$\mathcal{T} = \left\{ \mathbf{x}(t) \,\middle|\, \mathbf{x} \in \mathsf{R}^r, t \in \mathsf{R}^+ \right\}, \mathcal{T} \subseteq \mathbf{SE}^3 = \mathsf{R}^3 \times \mathbf{SO}^3$$

The vectors $\mathbf{x} = \mathbf{x}_n^0 = [\,\mathbf{p}\ \phi\,]^{\mathrm{T}}$, $n =$ no. of d.o.f. express the location of the nth coordinate frame (x_n, y_n, z_n), attached to the end-effector, relative to the world frame (x_0, y_0, z_0) attached to the base of the robot, $\mathbf{p} \in \mathsf{R}^3$ specifies the coordinates of the origin of the task frame (or end-effector frame), whereas the current orientation ϕ of the task frame is described either by the rotation matrix $\mathbf{R}$—a member of the special orthogonal group $\mathbf{SO}^3$, or minimally by a set of 3 Euler angles (in the sequel, the *yaw*, *pitch* and *roll* angles will be considered). If the tool is a single rigid body moving arbitrarily in the Cartesian 3D workspace, then $\mathcal{T} = \mathbf{SE}^3 = \mathsf{R}^3 \times \mathbf{SO}^3$, $m = 6$.

Because on one hand robotic tasks are specified with respect to one or more coordinate frames, and on the other hand visual servoing of robots makes intensive

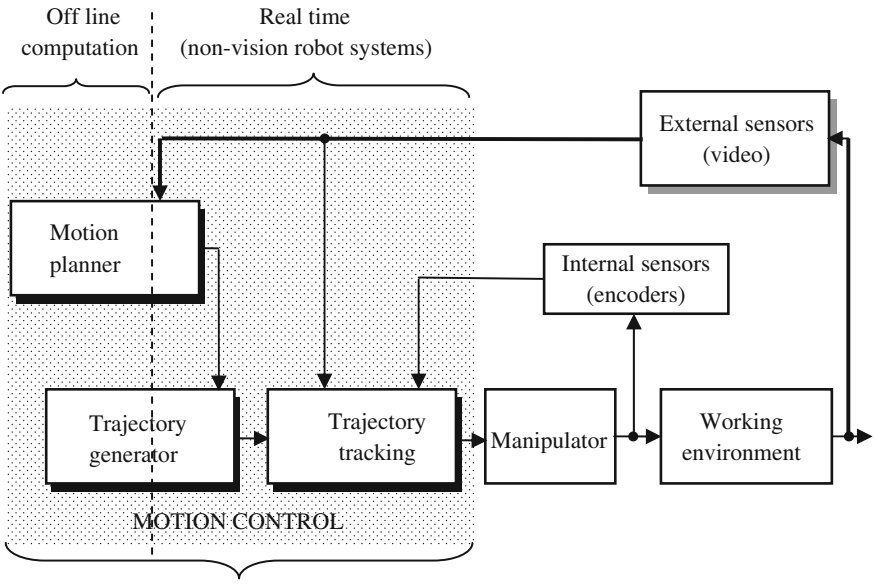

Fig. 1 Functional architecture for the global robot motion planning and control problem

use of a number of specific, additional coordinate frames, *coordinate transformations* are used in motion planning and tracking as a generalisation of *poses* to express relative locations between such frames of interest [2, 3].

Coordinate transformations must be often composed in the stage of motion planning and tracking, off line or at run time, to obtain the desired pose of the end-effector. Assuming that we are given the coordinate transformations $\mathbf{x}_{vis}^{0}$ and $\mathbf{x}_{obj}^{vis}$ expressing respectively the location of the coordinate frame (x_{vis}, y_{vis})attached to the image plane relative to the world frame (x_0, y_0, z_0) in the base of the robot, and the location of the frame (x_{obj}, y_{obj}) attached to an object relative to (x_{vis}, y_{vis}), then the coordinates $\mathbf{M}^{obj}$ of a point in the object frame can be expressed in the world frame by the composition rule (:):

$$\mathbf{M}^{0} = \mathbf{x}_{vis}^{0}[\mathbf{x}_{obj}^{vis}[\mathbf{M}^{obj}]] = (\mathbf{x}_{vis}^{0} : \mathbf{x}_{obj}^{vis})[\mathbf{M}^{obj}] = \mathbf{x}_{obj}^{0}[\mathbf{M}^{obj}]$$

The associated relative rotation matrix and translation are given by $\mathbf{R}_{obj}^{0} = \mathbf{R}_{vis}^{0}\mathbf{R}_{obj}^{vis}$, $\mathbf{p}_{obj}^{0} = \mathbf{R}_{vis}^{0}\mathbf{p}_{obj}^{vis} + \mathbf{p}_{vis}^{0}$. In the V+ structured robot programming environment, the *simple* transformations: `to.cam[cam]`—available from camera-robot calibration, and `vis.loc`—the object location computed at run time, stand respectively for $\mathbf{x}_{vis}^{0}$ and $\mathbf{x}_{obj}^{vis}$; the object-attached frame is related to the world frame by the *composed* transformation `obj.loc` ← `to.cam[cam]:vis.loc`

During *motion planning* stage, the desired paths are generated without timing information, i.e., without specifying the velocity and the acceleration along the path. Of primary concern is the definition of *collision-free paths* in the workspace. A secondary objective may be included, for example the optimization of some cost functions like: minimization of the total travel time or distance, keeping as low as possible changes in direction, continuity of velocity, etc. [4].

The trajectory planner (generator) parameterises the end-effector path *directly in the task space* either as a curve in $\mathbf{SE}^3$, or in R^6 when a minimal Euler representation is used for $\mathbf{SO}^3$. The trajectory planner may also compute a trajectory for the individual joints of the manipulator as a curve in the *configuration space* $C = \{\mathbf{q}(t) \,|\, \mathbf{q} \in \mathsf{R}^n, t \in \mathsf{Z}^+, n = \text{ no. of d.o.f.}\}$.

The *trajectory planner* TP, represented as block—and connection diagram in Fig. 2, is a software module, component of the basic software system of the robot controller, being characterised as follows:

1. The *inputs* to the TP are the path description and constraints, and the constraints imposed by the manipulator's dynamics.
2. The *output* from the TP is the joint—or end-effector trajectory data, expressed as a discrete time sequence of the values which must be attained by the position, velocity and acceleration computed respectively in the *configuration space* $\mathbf{q} \in C$ or in the *task space* $\mathbf{x} = (\mathbf{p}, \phi) \in \mathsf{R}^r$, from the initial to the final pose.
3. The *trajectory planning task* is executed by the TP in one of the two following modes:

   - Assuming that a *set of constraints* (e.g. continuity or smoothness) on position, velocity and acceleration of the manipulator's joint variables has been *explicitly*

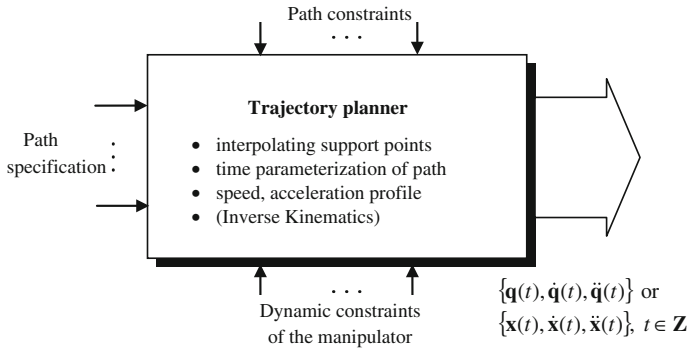

Fig. 2 Trajectory planner task and I/O representation

specified at selected joint configurations (called support—or *interpolating points*) along the trajectory, the TP selects then a parameterised trajectory from *a class of polynomial functions* in the total travelling time interval, which interpolates and satisfies the imposed constraints at the support points.

- A *path* that the end-effector must traverse *is explicitly specified* by *an analytical function* (e.g. a 3D straight-line path, a 2D circular-arc path in Cartesian coordinates or any computed curve), and the TP adds a time law to compute a trajectory that approximates the desired path either in joint coordinates or in Cartesian coordinates.

In the first mode, the constraint specification and the planning of the manipulator trajectory are performed in *joint coordinates*. In the second mode, the path constraints are specified in *Cartesian coordinates*, and the joint actuators are servoed in joint coordinates.

To compute a joint-space trajectory, a given end-effector path must be transformed into a joint-space path via the Inverse Kinematics (IK) mapping. Due to the difficulty of computing on line this mapping, the usual approach is to compute a discrete set of joint vectors along the end-effector path (joint support vectors), and then to interpolate in joint space between these support points in order to complete the joint-space trajectory. Common approaches to trajectory interpolation include polynomial spline interpolation using trapezoidal velocity profiles and time laws of blended polynomial type, cubic polynomial trajectories, or trajectories generated by reference models [5].

2 The Trajectory Generation Problem in Robot Motion Control

A path can be defined either in the joint space or in the operational space. Usually, the latter is preferred since it allows:

- a *natural description of the task* the manipulator has to do,

- a *simple description of the path constraints*—these are due to regions of the workspace which are forbidden to the manipulator (e.g. due to the presence of obstacles), and
- a *direct knowledge of the pose* of the end-effector in the workspace [6].

A geometric path cannot be fully specified by the user due to complexity reasons. Typically, a reduced number of parameters are specified, such as: final points, possible intermediate points, geometric primitives interpolating the points. Also, the *motion time law* is not typically specified at each point of the geometric path, but rather it regards: the total trajectory time, the constraints on the maximum velocities and accelerations or the eventual assignment of velocity and acceleration at some points of particular interest.

This section presents algorithms and implementing solutions for operational-space and joint-space and motion planning. Real-time computational aspects and performances are analysed.

2.1 Joint-Space Trajectory Planning

For this type of trajectory planning, the time history of all joint variables and of their first two derivatives is planned to describe the desired motion of the manipulator. Planning in the joint space has the following advantages:

- the trajectory is planned directly, in terms of the controlled joint variables $\mathbf{q}(t)$ during motion execution;
- the trajectory planning can be done nearly in real time;
- the joint trajectories are planned with a reasonable computational effort.

The main disadvantage is the difficulty in determining the locations of the various links and of the end-effector in the operational space, a condition which is usually required to guarantee obstacle avoidance along the trajectory.

The global algorithm for generating joint-trajectory set points is given next:

$t = t_0$;
loop: wait for next control interval;
$t = t + \Delta t$;
Update the trajectory planner $\mathbf{tp}(t) \rightarrow$ compute the necessary joint posture of the manipulator: $\{\mathbf{q}(t), \quad \dot{\mathbf{q}}(t), \quad \ddot{\mathbf{q}}(t)\}$ at time t ;
if $t = t_{\text{final}}$ exit;
else go to ***loop***.

Four constraints are imposed to the planned joint-space trajectory:

1. The trajectory set points must be non-iteratively readily calculable.
2. Intermediate points must be evaluated in a deterministic mode.

3. The continuity of the joint position and its first two time derivatives must be guaranteed so that the planned joint trajectory is smooth.
4. Extraneous motions must be avoided.

The constraints 1–4 for the planned trajectory will be satisfied if the *time history* of the joint variables can be specified by *polynomial sequences*.

Robot controllers use *electronic gearing* in the joint-space trajectory generator in order to synchronize the movement of one or more slave axes to the movement of a master device, which can be an encoder, A/DC, or the trajectory of another axis, e.g. the robot's leading axis which must execute the longest displacement.

2.2 Operational-Space Trajectory Planning

The general case of Cartesian-space planning is considered, for which the global algorithm is given below:

$t = t_0$;
loop: wait for next control interval;
$t = t + \Delta t$;
Update the operational hand planner $\mathbf{TP}(t) \rightarrow$ compute the necessary position and orientation of the end-effector: $\{\mathbf{p}(t), \quad \phi(t), \quad \dot{\mathbf{p}}(t), \omega(t)\}$ in the operational space at current control time interval t ;
Compute the closed IK joint solution – CIKS, IK[$\mathbf{TP}(t)$], corresponding to $\mathbf{TP}(t)$;
if $t = t_{\text{final}}$ exit;
else go to ***loop***.

In general, task-space planning is done in two steps:

STEP 1: Generating or selecting the *set of support points in operational coordinates* according to some rules, along the operational path
STEP 2: Specifying a *class of functions* to link the support points defined in STEP 1 (or to approximate the path segments) according to some criteria. The criteria which are chosen are often dictated by the control algorithm following the trajectory planning, which tracks the desired path.

There are two approaches which can be used for achieving STEP 2:

1. The ***operational space***—*oriented* approach: support points are generated along the task path in operational coordinates. Then, the TP interpolates in operational space between these support points and adds the time law expressed in terms of the desired speed and acceleration profiles. The result will be the discrete

time sequence of values that must be attained by the end-effector's position and velocity computed in the task space, i.e. *the trajectory in the task space*.

Next, the task-space trajectory is converted into the corresponding *joint-space trajectory*, by applying for inverse kinematics computation. Several techniques can be used to this purpose:

- The kinematics inversion using the Jacobean pseudo-inverse $\mathbf{J}^{+}$ or the Jacobean transpose $\mathbf{J}^{\mathrm{T}}$ [1, 7];
- The resolved motion rate control (RMRC) algorithm in the form:

$$\delta \mathbf{q}_c(t) = \mathbf{J}^{-1}(\mathbf{q}_c(t))\delta \mathbf{x}_c(t)$$

where:

$$\delta \mathbf{q}_c(t) = \mathbf{q}(t_{k+1}) - \mathbf{q}(t_k), \quad \delta \mathbf{q}_c(t) = I_{\mathrm{OS}}(\mathbf{x}_d(t_{k+1}) - \mathbf{dk}(\mathbf{q}(t_k))$$

This corresponds to an incremental IK task space computation I_{OS}, $\delta \mathbf{x}_c(t)$ being the incremental displacement along the operational path, and $\mathbf{dk}(\mathbf{q})$ is the time function that computes the Direct Kinematics model. Hence it can be observed that the two time-consuming computing tasks:

– operational path interpolation between support points, and
– conversion of the task-space trajectory to a joint-space trajectory

are performed *incrementally* with arguments representing relative position and speed values. This will consequently reduce the computation time and augment the bandwidth of the TP (Fig. 3).

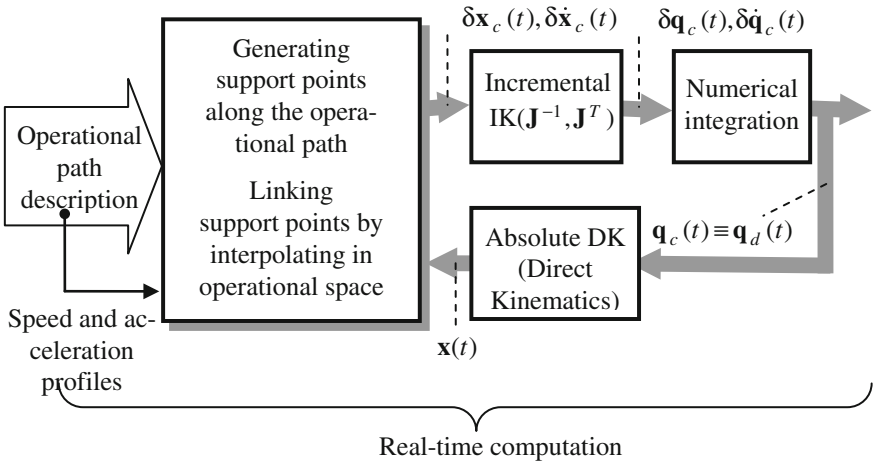

Fig. 3 Linking support points by interpolating in operational space

Moreover, in order to additionally reduce the computing time, truncation of results and approximations like $\sin\theta \approx 0, \cos\theta \approx 1 - \theta^2/2$ for $|\theta| < \varepsilon$ with ε a small, positive quantity, or reduced-order series development are accepted for the IK computation task. This is because any induced errors will be compensated by the DK task, placed on the feedback path and operating with the absolute values of the argument "$\mathbf{q}_c$—the computed joint configuration on the operational path" [5, 8]. In the case of linear interpolation, the resulting output trajectory generated by the TP is a *piecewise straight line* in the task space.

Attention must be paid as IK transformations do not produce unique solutions; in addition, if the manipulator dynamics is included in the trajectory planning, then path constraints will be specified in operational coordinates, while physical constraints such as force, torque, velocity and acceleration limits of each joint motor will be bounded in joint coordinates.

2. The ***joint space****—oriented* approach: *converts* first the support points that have been defined along the operational path into their corresponding joint coordinates, and the uses *low-degree polynomial functions* to interpolate between these converted support points (Fig. 4). Figures 3 and 4 represent two approaches used for interpolating between support points generated along the operational path.

If the TP must generate a linear trajectory in the task space, the support points will be on this linear path, but the linear joint-space interpolation between support points will produce a final output trajectory which is a non–piecewise straight line in the task space. According to the maximum allowed deviation in position of the planned trajectory with respect to the ideal, linear one in the task space, a certain number of support points will be defined on the operational path. The smaller the admitted deviation, the larger the number of support points to be defined on the operational path. This second approach is widely used, because of its reduced computational effort.

In the **trajectory tracking** stage, the computed reference trajectory is input to the motion controller, whose function is to determine the end-effector to *track the given trajectory as close as possible*. The trajectory tracking task is executed in real time

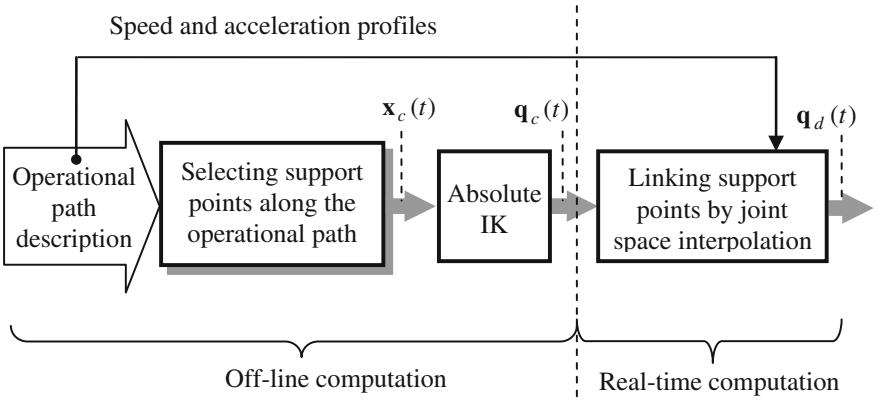

Fig. 4 Linking support points by interpolating in joint space

by the motion controller and consists in computing the time history of *joint control inputs* **u**, i.e. the vector of control voltages for the *n* axes' servomotors.

Task description is in most cases expressed in the *m*-dimensional operational space (with a particular minimal representation for the end-effector orientation), whereas control inputs (control velocities or forces/torques for the joint actuators) are generated in the *n*-dimensional joint space. Consequently, two types of motion control schemes have been thought of: with *joint-space* trajectory tracking and with *operational-space* trajectory tracking.

3 The Taxonomy of Visual Robot Servoing

The AI approach to intelligent robot automation is best characterised as the attempt to provide a robot with a symbolic representation of its environment and of its own actions, to be exploited by some kind of inference procedure. In this field, the main contributions of AI have been significant in two directions [9–11]:

- *perception*, with particular regard to object recognition and locating through vision;
- *planning*, i.e. the automatic construction of a sequence of actions capable to achieve a predefined goal.

The *behavioural intelligence* of a robotic system refers to the following properties:

1. *Flexibility*: in different situations, the robot controller is able to produce appropriately different behavioural patterns in pursuit of different goals.
2. *Robustness*: the robotic system can absorb and neutralise the effects of incomplete and noisy information and of limited changes in the environment's structure and dynamics.
3. *Adaptiveness*: the ability of the robotic system to alter behaviour significantly in response to radical changes in the environment.

Robot-vision systems use intelligent image processing to detect, recognize or track object features and act in consequence to plan and guide the motion of the robot. The chapter introduces the Look-and-Move approach for guidance vision (visually planning the robot's motion—the industry solution), see Fig. 5.

This is a *hierarchical* motion control structure, with the vision processor providing (planning) set-points as references to the robot's joint-level controller—thus using joint data feedback to internally stabilise the robot. This structure leads to an *interlaced look-and-move* control scheme, where motion tracking and image processing are *pipelined* as follows:

- while a motion segment is executed, no image is acquired and processed, and
- while an image is taken and treated according to the specific needs of a robot task, the motion controller does not start generating a trajectory and tracking it.

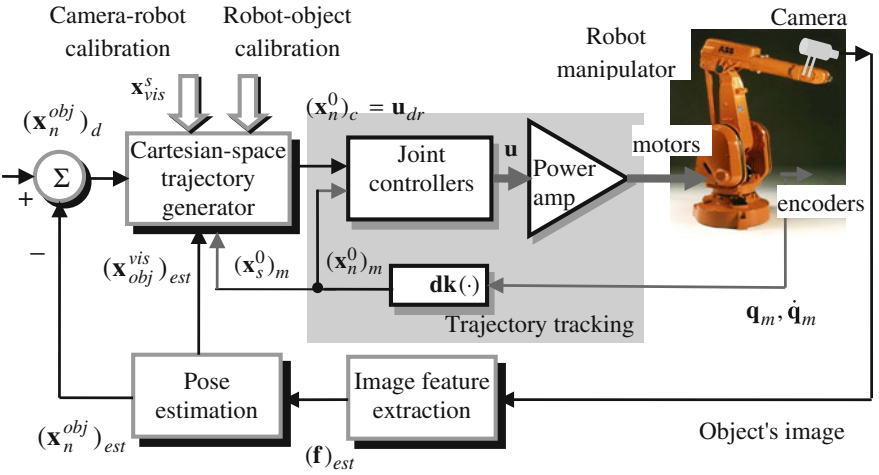

Fig. 5 Position-based look-and-move visual servoing architecture for object tracking

It can be observed that, whereas the global robotic system operates in an *open loop structure at motion control level*, it is subject to a *closed loop control at the global task level*.

Position-based look-and-move control is further discussed in this section. As described in Fig. 5, features are extracted from the image and used to estimate the pose $\hat{\mathbf{x}}^{vis}_{obj} = (\mathbf{x}^{vis}_{obj})_{est}$ of the target (object, point) with respect to the camera. Using these values, an error between the current estimated and the desired pose of the robot, $(\mathbf{x}^{vis}_{obj})_d$ is defined in the task space $\mathcal{T}$. Thus, position-based control neatly separates the control actions, i.e. the computation of the feedback signal $(\mathbf{x}^0_s)_m = \mathbf{dk}(\mathbf{q}_m, s), n-3 \leq s \leq n$ using the direct kinematics model $\mathbf{dk}(\cdot)$ of the robot manipulator, from the estimation problem involved in computing position or pose $\hat{\mathbf{x}}^{vis}_{obj}$ from visual data $(\mathbf{f})_{est}$.

A visual positioning task is expressed by an error function $\mathbf{E} : \mathcal{T} \rightarrow \mathsf{R}^m$. This function is referred to as the *virtual kinematic error function* VKE. A positioning task is fulfilled when the end-effector has been moved in pose $\mathbf{x}_n = \mathbf{x}^0_n$ if $\mathbf{E}(\mathbf{x}_n) = \mathbf{0}$.

Once a suitable VKE function defined and its parameters instantiated from visual data, a compensator can be designed that reduces the value of the VKE function to zero. This compensator computes at every sampling time instant the necessary end-effector position $(\mathbf{x}_n)_c$ that is sent as dynamic reference to the joint-space (or operational-space) motion tracking controller [1]. Since the VKE functions are defined usually in the Cartesian space, it is common sense to develop the compensator's control law through geometric insight.

4 Guidance Vision for Robot Motion Planning

The problem of visual feature tracking for robot motion planning and object access control will be further presented for two types of working environments: (1) fixed scene, e.g. workstation, storage, ASRS, and (2) mobile scene, e.g. conveyor belts [12, 13].

4.1 Open, Vision-Based Robot Motion Planning for Fixed Scene Foreground

An open, vision-based robot motion planning and control method and implementing solution is presented in this section. The method allows using any general purpose machine vision system (here an industrial camera with c-mount and AdeptSight software) with any type of industrial robot controller (here ABB), with a proper interfacing (Ethernet or serial communication).

In order to be used, a camera calibration is needed (which is provided by vision any image processing library based on a calibration pattern), and also a robot-camera calibration (which must be done manually by the robot technician); the models of the objects to be accessed by the robot and the robot-object (class) grasping will be off-line taught for collision free motion at execution time.

In industrial applications of position-based dynamic look-and-move control structures, the robot-vision system works in most cases with off line learned objects which can be visually recognised and located at run time [14, 15]. It becomes thus possible:

- to recover the object's pose, $\hat{\mathbf{x}}_{obj}$, relative to the base frame of the robot, from the direct estimate $\hat{\mathbf{x}}_{obj}^{vis}$ of the object's pose in the vision frame and by composing it with the camera-robot calibration estimate $\hat{\mathbf{x}}_{vis}$;
- to define stationing points $\mathbf{S}^{obj}$ on the object's image, relative to a suitable object-attached frame (x_{obj}, y_{obj}).

Figure 6 shows a fixed camera configuration and related camera-robot transformations; this is an *endpoint open-loop* (EOL) system that only observe the target object to guide the robot's motion for grasping it.

The physical camera is related to the base coordinate system of the robot by the time-invariant pose evaluated a single time during an interactive off line camera-robot calibration session, and to the object in the scene by. The camera image of the object is independent of the robot motion (unless the target is the end-effector itself, described for example by image feature of the gripper's fingerprints projected onto the image plane). The pose is computed at run time, and involves the search, recognition and locating of image features(s) on the object of interest [16–18].

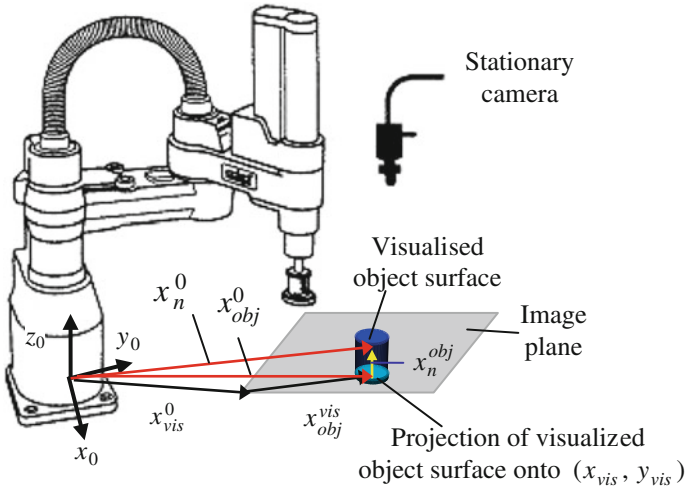

Fig. 6 Stationary camera configuration and related camera—robot relative transformations x_{obj}^0, x_n^0 respectively for the *feature tracking* and *feature tracking for object grasping* tasks

For object grasping, the image features must unambiguously describe the entire object for its successful identification and locating at run time. In addition, the pose of the gripper, relative to the frame attached to the object in its current location, is required.

For a *stationary camera*, the relationship between these poses is:

$$\mathbf{x}_n^0 = \mathbf{x}_{vis}^0 : \mathbf{x}_{obj}^{vis} : \mathbf{x}_n^{obj}, \text{ for feature tracking for object grasping.}$$

Assuming a random part presentation in the robot workstation, the object's pose relative to a (unique) camera frame, $\hat{\mathbf{x}}_{obj\_1}^{vis}$ will be estimated at run time in a *first stage* in terms of the following image feature parameters:

- x_C, y_C: coordinates of the centre of mass C of the 2D projection of the object's visualised surface onto the image plane (x_{vis}, y_{vis});
- $orient = \angle(\text{MIA}, x_{vis})$: orientation angle of the object.

The object-attached frame (x_{obj_1}, y_{obj_1}) has the origin in **C** and the abscissa $x_{obj\_1} \equiv$ MIA, where MIA stands for the object's Minimum Inertia Axis (Fig. 7).

To move the robot to grasp objects of a certain class always in the same way, irrespective of their location in the robot scene, the desired (unique) pose of the gripper, $\mathbf{x}_{n^*}^{obj}$, relative to the object-attached frame must be a priori learned.

Let us denote by **G** the projection of the end-tip point **T**, the origin of the gripper-attached frame (x_n, y_n, z_n), onto the image plane: $\mathbf{G} = \text{projl}_{(x_{vis}, y_{vis})}\{\mathbf{T}\}$.

For a desired grasping style, $\mathbf{G}^{obj\_1}$ is a stationing point in the object's coordinates (x_{obj_1}, y_{obj_1}), irrespective of the current position and orientation of the object. Its

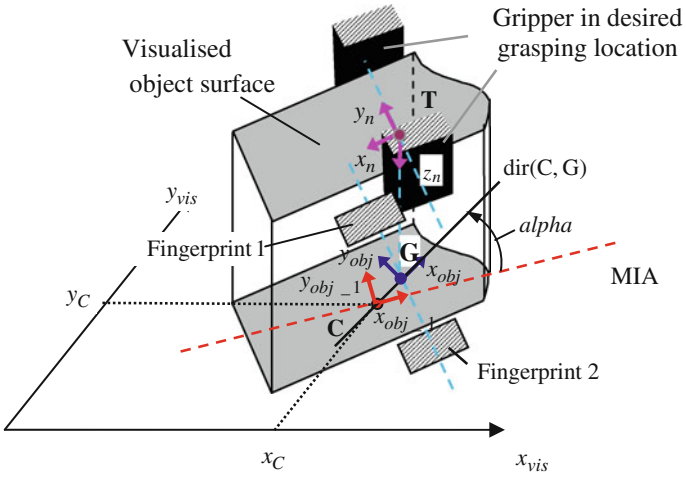

Fig. 7 Definition of the object-attached coordinate frame

coordinates are: $x_G = d_{CG} \cdot \cos(alpha)$; $y_G = d_{CG} \cdot \sin(alpha)$, where $d_{CG} =$ dist(**C**, **G**), and $alpha = \angle(\text{dir}(\mathbf{C}, \mathbf{G}),\ \text{MIA})$ measured CCW from the Minimum Inertia Axis MIA) to dir(**C**, **G**). In a second stage, the object-attached frame will be shifted to origin **G**, by a translation of distance d_{CG} along dir(MIA) followed by a rotation of angle *alpha* about the normal in **C** to the image plane, as represented in Fig. 7.

Given an object pose, $\mathbf{x}_{obj}^{vis}$, estimated visually at run time, and assuming that the object was recognised as a member of that class for which a relative grasping pose $\mathbf{x}_{n^*}^{obj}$ was *a priori* learned using a stationary camera calibrated to the robot base frame by $\mathbf{x}_{vis}$, then the positioning error can be defined by the VKE function

$$\mathbf{E}(\mathbf{x}_n; \tilde{\mathbf{x}}_{n^*}^{obj}, \hat{\mathbf{x}}_{obj}^{vis}, \hat{\mathbf{x}}_{vis}) = \mathbf{x}_{n^*}^{n} = \hat{\mathbf{x}}_0^n : \hat{\mathbf{x}}_{vis} : \hat{\mathbf{x}}_{obj}^{vis} : \tilde{\mathbf{x}}_{n^*}^{obj},$$

where:

$$\tilde{\mathbf{x}}_{n^*}^{obj} = \begin{cases} \mathbf{x}_{n^*}^{obj} \text{ a priori known from learning,} & \textit{particular} \text{ “grasping style”} \\ \hat{\mathbf{x}}_{n^*}^{obj} \text{ visually updated at run time,} & \textit{general} \text{ “grasping style"} \end{cases}.$$

With an EOL system, $\hat{\mathbf{x}}_0^n = \text{inverse}(\hat{\mathbf{x}}_n^0)$ will be dynamically updated by the trajectory generator to bring to zero the positioning error $\mathbf{x}_{n^*}^n$. This can be simply done applying for an IK-based *Resolved Motion Rate Control* algorithm.

The closed-loop servo control uses the visually estimated pose of the object, $\hat{\mathbf{x}}_{obj}^{vis}$, the estimated camera-robot calibration pose $\hat{\mathbf{x}}_{vis}$, and assumes that reduced-error direct kinematics ($\hat{\mathbf{x}}_n^0$)—and inverse kinematics ($\hat{\mathbf{x}}_0^n$) models are available. As for the imposed grasping pose, for a priori unknown object location in the scene, some components in $\hat{\mathbf{x}}_{n^*}^{obj}$ must be estimated at run time whenever the “style" in which the object will be grasped is *general*, i.e. such that $\mathbf{G} \not\equiv \mathbf{C}$ and $\mathbf{G} \notin$ MIA [19–21].

For object access and handling using vision, the problem is reduced to expressing the object position in the image relative to the robot base. This is done in the *robot-camera calibration* session, the result of which is a relative transformation expressing the position and orientation of the vision frame relative to the robot base. Once the calibration is executed, robot points will be computed relative to the position and orientation of the vision-attached frame, and the robot motion planning follows the procedure described in Sect. 2.

The robot-camera calibration procedure requires the usage of an object that will be handled by the robot; during the execution of the procedure the robot will move the object to different locations and will acquire pictures, generating a set of pairs of descriptions of the object's location: (a) from the camera and (b) from the joint encoders. The solution of this set of equations will describe the camera's field of view location relative to the robot base.

For testing purpose an AdeptSight system and ABB robot manipulator were used, the robot-vision calibration process and the training of the object grasping model have been integrated in a single procedure. The procedure consists in four human-robot interactive steps where the robot grasps the object and places it different positions in the workspace for image acquisition and processing [2]:

The calibration object is placed in the workspace and grasped by the robot and then released (position P1), after which the robot clears the vision plane and the object's position in the vision plane is computed by the AdeptSight library. The point P1 is the point which will be used to express all the positions of the objects in the image. For example a position of an object will be computed as Po where Po is P1 shifted with a set of offsets (for translations on X and Y and rotation on the Z axis). The position of the object in point P1 is also computed in the vision plane, having the coordinates $P1_{Xv}$, $P1_{Yv}$ (the position of the coordinate system attached to the object model) [22, 23].

In the second step the robot grasps the object and places it in the same position, but rotated with 180° (point $P1'$), in the vision coordinates $P1'_{Xv}$, $P1'_{Yv}$. By comparing the position of the coordinate system of the object in these positions the system can compute the position of the mass centre of the object (the mass centre of the model relative to the grasping point). In this case the grasping point is located in the image on the middle of the segment $[P1, P1']$ (see Fig. 8).

$$P_{gvis} \begin{cases} P_{gvisx} = \min(P1_x, P1'_x) + \left| P1_x - P1'_x \right| \\ P_{gvisy} = \min(P1_y, P1'_y) + \left| P1_y - P1'_y \right| \end{cases},$$

where P_{gvis} is the grasping point in the vision workspace.

Next the object is placed in a position P2 which is trained relative to the position P1 shifted with 100 mm on X axis of the base coordinate system of the robot.

In the final step the robot places the object in the position P3 which is trained relative to the position P1 shifted with 100 mm on Y axis of the base coordinate system of the robot. By knowing the correspondence robot-point—image-point, the system can compute now the orientation of the vision plane relative to the robot base

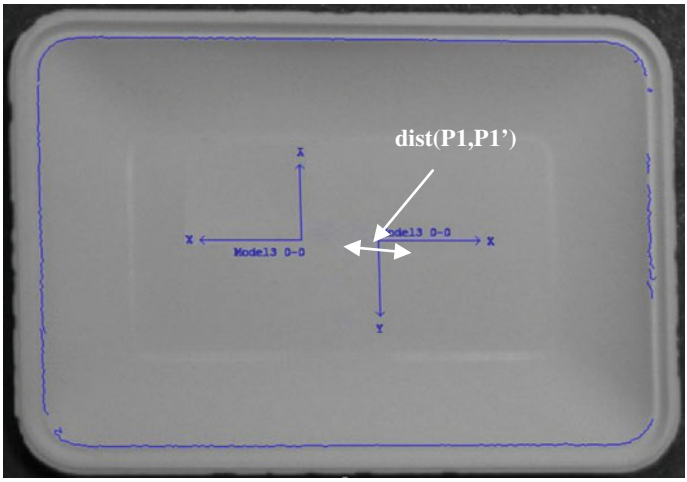

Fig. 8 The relationship *robot point—vision point*

coordinate system, and also the distance which the robot must cover to reach an object which is placed at a certain distance from the initial point P1 in the image plane.

This can be expressed as follows: for 100 mm travelling length along the X coordinate system (base coordinate system) the object moves in the image $P2_x - P1_x$ along the X_{vis} axis, and $P2_y - P1_y$ along Y_{vis}; the same travelling length on the Y coordinate system generates $P3_x - P1_x$ on X_{vis} axis, and $P3_y - P1_y$ on Y_{vis}, in the vision workspace. It results also that the vision system is rotated with the angle:

$$\alpha = a\tan 2(P2_Y - P1_Y, P2_X - P1_X)$$

toward the base coordinate system. Hence for an object which is recognized in the image at the location P_v, the object will be grasped at the coordinates:

$$\begin{aligned} P_x &= P_{G\_x} + \sqrt{(P1_{\_x} - P_{v\_x})^2 + (P1_{\_x} - P_{v\_x})^2} \\ &\quad \cdot \cos(\alpha + a\tan 2(P_{v\_y} - P1_{\_y}, P_{v\_x} - P1_{\_x}) \\ P_y &= P_{G\_y} + \sqrt{(P1_{\_x} - P_{v\_x})^2 + (P1_{\_x} - P_{v\_x})^2} \\ &\quad \cdot \sin(\alpha + a\tan 2(P_{v\_y} - P1_{\_y}, P_{v\_x} - P1_{\_x}) \\ P_{rot} &= P_{G\_rot} + (P_{v\_rot} - P1_{\_rot}) \end{aligned}$$

where P_x, P_y, P_{rot} are the position coordinates and the rotation of the grasping point of the object which was located in the vision workspace at the location P_v (P_{v_x}, P_{v_y}, P_{v_rot}); P_G (P_{G_x}, P_{G_y}, P_{G_rot}) is the grasping point (in the object's centre of the mass in the base coordinates system) for the object located in the image in P1 ($P1_{\_x}$, $P1_{\_y}$, $P1_{\_rot}$).

After the calibration is executed, the object model must be trained; this stage involves object edges processing in order to obtain the geometrical model of the object. The grasping position must be also trained in order to validate a collision free point for accessing the object. The grasping position (for grasping validation) is defined by two or more rectangular areas placed around the object and linked to the object frame. These areas represent the projections of the gripper fingerprints on the image plane and by processing the image colour inside these areas the program detects the presence of obstacles and can invalidate the grasping position.

These three pieces of information are used for robot motion planning; first the location of the field of view is used by extracting it from the calibration data, then the location of the object in the field of view is computed (*online*) using the object model and in the last stage the action of grasping the object is validated by using the grasping model and collision free tests. Experimental results validating the proposed solution are shown from a robotized ceramic production line (Fig. 9).

The experimental application runs two communications threads: a TCP/IP server and a serial communication thread. Both threads have the same role, they are listening and if they receive an acquisition request, they initialize the execution of the AdeptSight sequence of tools (the vision program), returning three numbers specifying the position and orientation of the plate (X, Y in mm, and the angle in degrees). The position and orientation is specified relative to the calibration object.

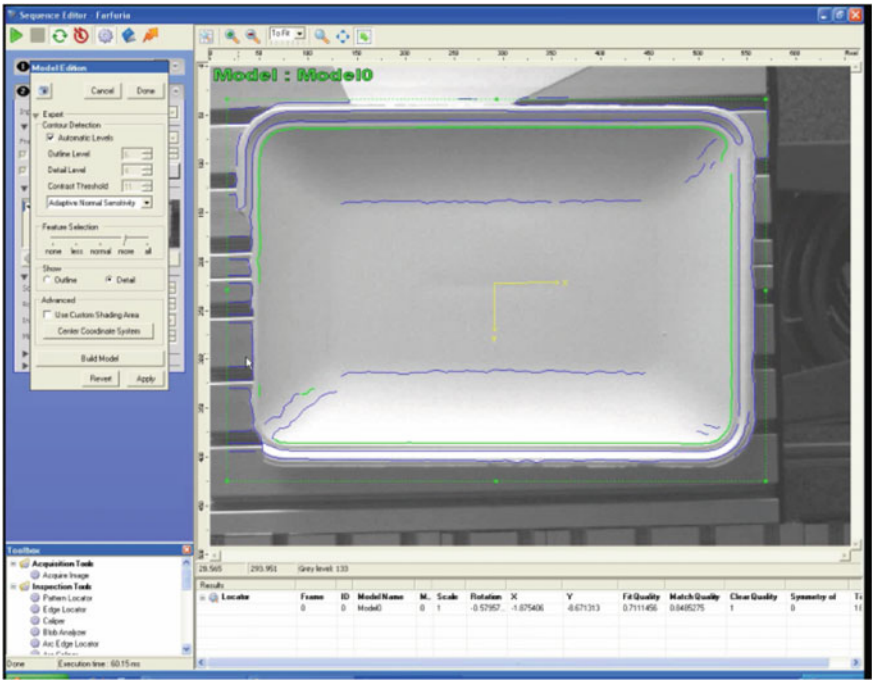

Fig. 9 Real-time locating a ceramic plate for robot motion planning and grasping control

The requests are sent as ASCII characters, and they are of two types (*i*—information for debugging or *r*—real requests); when the vision server receive a request the vision sequence is executed, the object is recognized based on its boundary contours, and the values (X, Y and rotation) relative to the initial grasping point (from the calibration procedure) are computed and sent to the ABB robot.

When the robot receives the three values, it shifts the initial grasping position (from the calibration) and grasps the plate. The following pseudo-code describes how the communication is integrated with the vision server [24]:

```
Open the communication channel (Serial line)
Clear the serial line buffer
Request object coordinates from vision
Read the data streams (X,Y coordinates and rotation)
Transform the coordinates from string to real
Request object coordinates from vision
Read the data streams (X,Y coordinates and rotation)
Transform the coordinates from string to real
/*In order to avoid problems caused by communication errors
the coordinates are sent twice and only if they are the
same at the destination then the position can be computed*/
Verify if the coordinates are the same
IF YES
     Compute the grasping position
     //The position is computed relative to a predefined
     //position p1
     Close the communication
ELSE
     Repeat the request
```

The presented image processing system, AdeptSight, is robust, offers generic robot-vision functions, and can be easily integrated with controllers of other industrial equipment (robots, measuring machines, ASRS, part feeders). AdeptSight allows a rapid development of visually planned applications, based on visual tools which can be combined and configured leading to sequences which can be executed from external C# applications.

4.2 Multitasking Robot Motion Planning for Object Tracking on Mobile Scenes

The problem of robot tracking objects of interest moving on conveyor belts and randomly entering the robot's dexterous space can be solved by integrating the following *devices* in a multitasking control structure, implemented on multiprocessor robot controllers:

- the robot manipulator, tracking a conveyor belt;
- the conveyor belt, driven at constant, regulated speed;
- the vision module, inspecting parts on the conveyor belt.

Conceptually, the problem is solved by defining a number of user tasks which attach two types of "robots": the *n*—d.o.f. manipulator grasping on-the-fly objects moving on the conveyor belt, and one $m \leq 3$—axis "robot" emulating the conveyor belt under vision control; *m* is the number of non-null projections of the conveyor belt displacement direction on the 3 axes of an orthonormal reference frame (e.g., defined in the belt tracking robot environment). These user tasks run concurrently with the internal system tasks of a multitasking belt tracking robot controller, which are responsible for trajectory generation, axis servoing and resources management [20].

In this respect, the minimum number of tasks to be defined for the tracking problem is equal to 3:

- Task 1: Dynamic re planning of the destination location (grasping the moving object) for the robot manipulator.
- Task 2: Continuously moving (driving) the *m*-axis vision belt. (e.g., $m = 1$)
- Task 3: Reading once the belt's location the very moment an object of interest has been recognised, located and its grasping estimated as collision-free, and then continuously until the object is effectively picked.

4.2.1 Tasks and Priorities for the Multitasking Robot Motion Planning Problem

Consider that each control system cycle of the robot is divided into 16 time slices of one millisecond, the time slices being numbered 0 through 15. A single occurrence of all 16 time slices is referred to as *a major cycle*. For a robot system, each of these cycles corresponds to one output from the trajectory generator to the digital servos. A number of user tasks, e.g. from 0 to 6, can be used and configured to suit the needs of specific applications. Tasks are normally assigned default time slices and priorities according to the current system configuration [5, 8].

An execution cycle is terminated when a STOP instruction is executed, a RETURN instruction is executed in the top-level program, or the last defined step of the program is encountered. Tasks are scheduled to run with a specified *priority* in one or more time slices. Tasks may have priorities from −1 to 64, and the priorities may be different in each time slice. The priority meanings are: 1–31 (normal user tasks); 32–62 (used by robot controller's device drivers and system tasks); 63 (used by trajectory generator); 64 (used by the servo).

4.2.2 Scheduling Program Execution Tasks with Simultaneous Belt Tracking

An analysis of the time slice and priority allocation for the system, and of default user tasks imposes several requirements for timing and priority assignment of tasks: *vision guided robot planning* ("object recognition and locating"), and *dynamical re planning of robot destination* ("robot tracking the belt") should always be configured on user

tasks 0 and/or 1, in "Look-and-Move" interlaced robot motion control applications, due to the continuous assignment of these two tasks, over the first 13 time slices, with high priorities [25].

Because vision guidance and motion re planning programs complete their computation in less than the 13 time slices (0–12), in order to give the chance to conveyor-associated tasks ("drive" the vision belt, "read" the current position of the vision belt) to provide the "robot tracking" programs with the necessary position update information earlier than the slice 13, and to the high-priority trajectory generation system task to effectively use this updates, a WAIT instruction should be inserted in the loop-type vision guidance and motion re planning programs of tasks 0 and/or 1.

All time slices are checked, wrapping around from slice 15 to slice 0 until the original slice is reached. If no runnable tasks are encountered, a null task executes. Whenever a 1 ms interval expires, the multitasking OS performs a similar search of the next time slice. If the next time slice does not contain a runnable task, execution of the current task continues. If more than one task in the same time slice has the same priority, they become part of a *round-robin scheduling group*. Programs that execute in continuous loops, like vision guidance and motion re planning for belt tracking, should generally execute a WAIT instruction occasionally (for example, *once through each loop execution*). This should not be done, however, if timing considerations for the tracking application preclude such execution delays in some stages of vision and motion processing [6, 26].

As previously stated, the problem of conveyor tracking with vision guiding for moving part identification and locating requires the definition of three *user tasks*, to which the following programs were associated:

1. Task 1: program "track" executes in this task, with robot 1 (e.g., SCARA) selected. This program has two main functions, carried out in a 2–*stage* sequence:

   STAGE 1: Continuous *checking* whether an object travelling on the conveyor belt (it will be called in the sequel *vision belt*) entered the field of view of the camera and the reachable workspace of the SCARA robot. If such an event occurs, the vision is activated to *identify* whether the object is of interest and to *locate* it. Processing on this stage terminates with the *computation of the end-effector's location* which would move the SCARA robot in the object picking location evaluated *once* by vision.

   STAGE 2: Continuously *re planning* the end-effector's location, computed when the object of interest was located by vision, by *consuming the belt position data* produced by encoder reads in the program "read" which executes on task 3, and by *dynamically altering the robot's target* in the current motion segment.

2. Task 2: program "drive" executes in this task, with robot 2 ((m = 1)-axis robot, i.e. the conveyor belt) selected. This program *moves the belt* in linear displacement increments, at a sufficiently high rate to provide a jerk-free, continuous belt motion. This program executes in stages 1 and 2 previously defined.

3. Task 3: program "read" executes in this task, with robot 2 selected. This program executes differently in the two stages of the application:

   STAGE 1: Executes *a single time* upon receiving an input signal ("la_reco", e.g. for "LA" objects of interest) from vision in task 1, confirming the recognition and successful locating of an "LA" part. In response, "drive" *reads the instantaneous belt position*, which from now on will be used as an offset for the position updates.

   STAGE 2: Continuously *reads the belt position,* upon a request ("info" in the example of the first case study) issued by "track" in task 1, when it starts its dynamic target re planning process.

From the three user tasks, the default priority assignment is maintained. This leads to the following priority analysis for a major cycle:

- Task 1 has the highest priority in time slices 0–12 (inclusively), with values of 19, 21, 9 and 11.
- Task 2 has the highest priority (20) in a single time slice: 13.
- Task 3 never detains a position of highest priority with respect to tasks 1 and 2.
- The three tasks become part of a round-robin group as follows:
  - tasks 2 and 3 in slices 0–12 inclusively,
  - tasks 1, 2 and 3 in slices 14 and 15.

Because tasks 2 and 3 are in more than one round-robin group on different slices, then all three tasks in the corresponding pairs of different slices appear to be in a big group. This property can cause, in general, a task to be run in a slice one does not expect; however, this risk is eliminated for task 1 in STAGE 2 since it will never be runnable in slices 14 and 15 (after generating a WAIT).

As for tasks 2 and 3, they cannot generate this risk in the remaining slices from 0–12, after "track" generates the WAIT, because they will switch continuously between them at the beginning of each new time slice.

As a result of the priority scan and scheduling, the programs in the three user tasks execute as follows:

- STAGE 1—vision is processing, the SCARA robot is not moving and no WAIT is issued by task 1 (Fig. 10):
- STAGE 2—*vision is not processing, the* SCARA *robot is moving and* WAIT *commands are issued in* task 1 *by the* "track" *program after each re planning of the end-effector's target destination within a V+ major cycle of* 16 ms:
  - Task 1 runs in slices $i - j, i \leq j, i \geq 0, j \leq 12$, (when it detains the highest priority), i.e., starting with the time moment when it is authorised to run by the highest-priority system tasks "trajectory generation" and "servo" (in slice *i*), and executing until it *accesses the position update* provided by task 3 *from the most recent belt encoder read*, *alters the last computed end-effector destination* and issues a WAIT (in slice *j*), to give the trajectory generator a chance to execute.

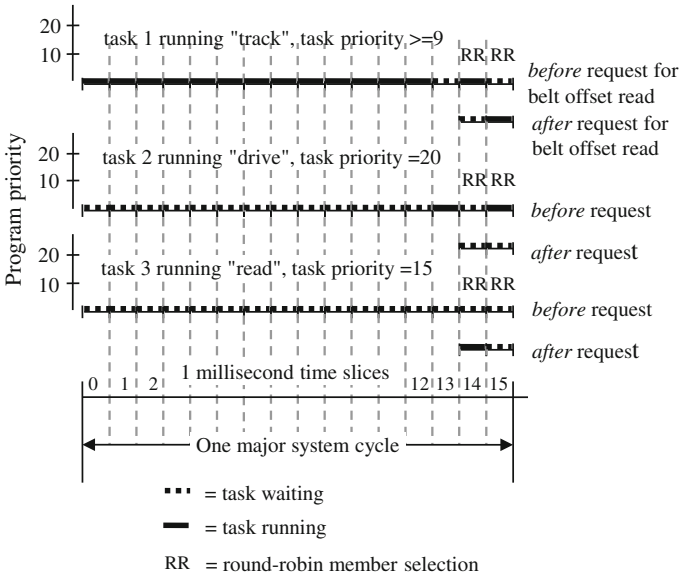

Fig. 10 Priority assignment and tasks running in STAGE 1 of vision guidance for motion planning in the belt tracking problem

– Task 2 runs: in slices $(j+1)-12$ switching alternatively with task 3 whenever it is selected as the member of the round-robin group following task 3 that run most recently, in slice 13 (it detains the highest priority), and in slice 15 (it is member of the round-robin group following task 3 that run more recently—in slice 14). Task 2 runs always exactly for 1 ms whenever selected, so that the round-robin group scanning authorises task 3 to run always at the beginning of the next time slice.
– Task 3 runs in slices $(j+1)-12$ switching alternatively with task 2 when-ever it is selected as the member of the round-robin group following task 2 that run most recently, and in slice 14 (it is member of the round-robin group following task 2 that run more recently—in slice 13). The task 3 runs, whenever selected, for less than 1 ms and issues a RELEASE "to anyone" command.

4.2.3 Dynamically Altering Belt Locations as Robot Motion References

The three previously discussed user tasks, when runnable and selected by the system's task scheduler, attach respectively the robots:

- Task 1: **robot 1**—a SCARA-type robot (e.g. Adept Cobra 600) is considered in this case
- Task 2, 3: **robot 2**—the "vision conveyor belt" of a flexible feeding system is considered.

Program "**track**" executing in task 1 has two distinct timing aspects: during STAGE 1, "track" waits first the occurrence of the on-off transition of a signal from the photocell, indicating that an object passed over the sensor and will enter the field of view of the camera. Then, after waiting for a period of time set up function of the belt's speed, "track" commands the vision system to acquire an image, identify an object of interest and locate it [27, 28].

During STAGE 2, "track" alters continuously, once per each major 16 ms system cycle, the target location of the end-effector, part.loc, that was computed (when one "LA"-part was located by vision and returned in the vis.loc transformation) by composing the following relative transformations (the ":" character stands for composition)

```
part.loc=to.cam[1]:vis.loc:grip.la
```

Here grip.la is the off line learned grasping transformation for the class of "LA" objects. The updating of the end-effector target location for picking-on-the-fly "LA" objects according to a predefined grasping style uses the V+ operation:

```
ALTER () Dx,Dy,Dz,Rx,Ry,Rz
```

which specifies the magnitude of the real-time path modification that is to be applied to the robot path during the next trajectory computation (`Dx,Dy,Dz/Rx,Ry,Rz` are the translations/ rotations respectively along the *X*, *Y*, *Z* axes).

This operation is executed by "track" in task 1 that is controlling the robot 1 (SCARA) in *alter mode*, enabled by the `ALTON` command. When *alter* mode is enabled, this instruction should be executed *once during each trajectory cycle*. The stopping decision is taken in "track" by using the `STATE (select)` function, which returns information about the state of robot 1 ("Motion stopped at planned location") selected by task 1 executing the ALTER loop. The ALTOFF operation was used to terminate real-time path-modification mode (alter mode) [3, 10, 14].

Program "**drive**" executing in task 2 has a unique timing aspect in both STAGES 1 and 2: when activated by the main program, it issues continuously motion commands for the individual joint number 1 of robot 2—the vision belt.

Program "**read**" executing in task 3 evaluates the current motion of robot 2—the vision belt along its single axis, in two different timing modes. During STAGE 1, upon receiving from task 1 the request `la_reco` (an instance of "LA" was recognised) to compute the belt's offset, reads the current robot 2 location and extracts the component along *Y*.

This invariant offset component, read when the "LA" was successfully located by vision and the grasping authorised as collision-free, will be further used in STAGE 2 to estimate the updates of the `y_off` motion, to alter the SCARA robot's target location along the *Y* axis.

The program below shows how the STATE function is used to stop the continuous updating of the end-effector's target location by altering at every major cycle the position along the *Y* axis. The altering loop will be exit *when motion stopped at planned location*, i.e. when the robot's gripper, moving to track the part

travelling on the conveyor belt, arrives in the imposed picking position relative to the moving part.

```
ALTON () 2    ;Enable altering mode

;The robot is commanded to move towards the grasping position
;computed when the object was VLOCATEd by vision.

MOVES part.loc
WHILE STATE(2)<>2 DO

;While the robot is far from the moving target (motion not
;completed at planned location...

      ALTER(),-pulse.to.mm*y_off

;Continuously alter the target grasping location

      WAIT

;Wait for the next major time cycle to give the trajectory
;generator a chance to execute

END
ALTOFF              ;Disable altering mode
CLOSEI              ;Robot picks the tracked object
DEPARTS             ;Robot exits the belt tracking mode
MOVES place

;Robot moves towards the fixed object-placing location place
```

In the example presented, the ALTOFF operation has been used to terminate real-time path-modification mode (alter mode). The instruction suspends program execution until any previous robot motion has been completed (similarly to a BREAK instruction), and then terminates real-time path-modification mode.

After *alter* mode terminates, the robot is left at a final location that reflects both the destination of the last robot motion and the total ALTER correction that has been applied [13, 17, 29].

The cooperation between the tasks on which run “track”, “drive” and “read” is shown in Fig. 11.

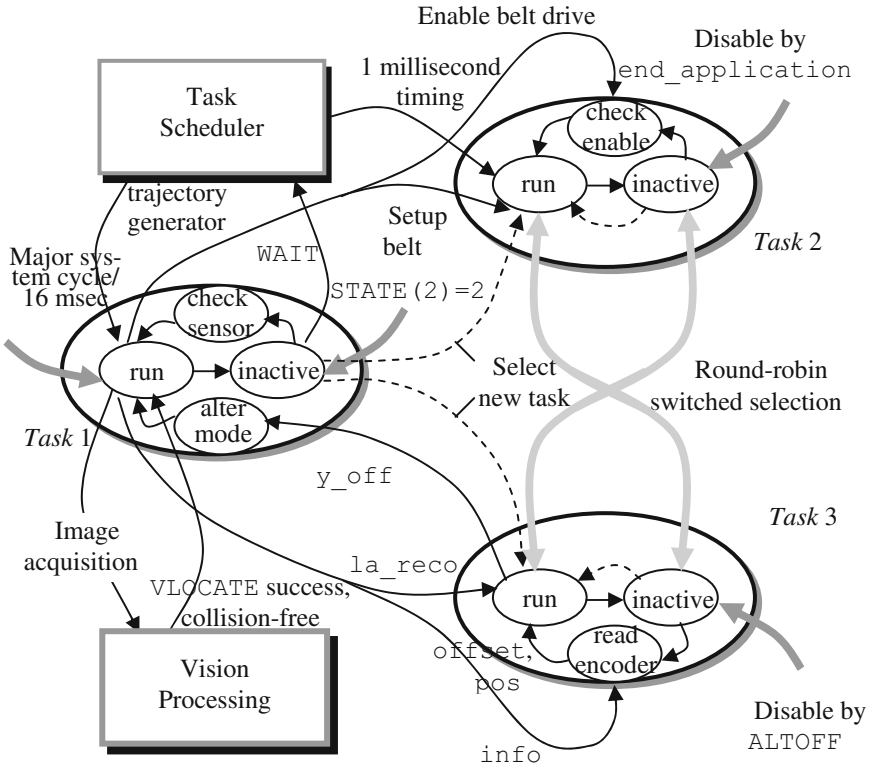

Fig. 11 Cooperation between tasks in the robot-vision belt tracking problem

5 Experimental Results and Conclusions

Visual robot motion planning allows relaxation of the numerous constraints which arise when setting up a manufacturing environment, as well as the need for high-precision material transportation and presentation devices, such as conveyors, vibrating bowls, a.o. The *look-and-move motion planning* methodology offers a robust solution to create workstations with components from different manufacturers: image sensors and cameras, vision software, robot manipulators, shop floor conveyors and other mechanical devices.

The open architecture system for vision-based robot motion planning application was developed in C# and managed the robot-vision communication and sequence execution. Also the camera-robot calibration procedure and the learning of the grasping model learning were developed in the same open system concept based on standard communication means [30, 31].

Figure 12 shows a screen capture of the open architecture robot-vision user application interface. The application consists in precision locating with AdeptSight of

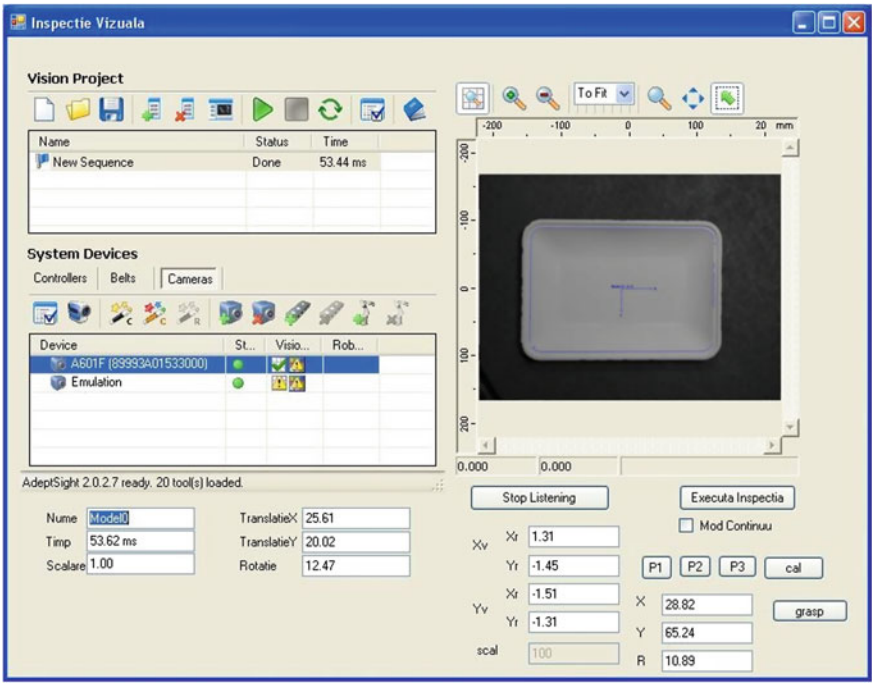

Fig. 12 Screen of the application interface for high-precision plate locating and vision-based robot motion planning for stationary plate grasping

ceramic plates travelling on a conveyor belt and guiding the motion of an ABB robot with help of the location data sent via standard communication channels and interfaces.

The motion control method presented above for robots picking on-the-fly objects on moving scenes was implemented in the V+ robot programming environment with AdeptSight vision extension, and tested on a robot vision platform containing one Adept Cobra 600 SCARA-type manipulator, a 3-belt flexible feeding conveyor Adept Flex Feeder 250 and a stationary, down looking matrix camera Panasonic GP MF 650 inspecting the vision belt with backlighting [10]. The vision belt on which parts are travelling and are viewed by a fixed, down looking camera was positioned parallel to the Y_0 axis of the manipulator, for a convenient robot access within a window of 460 mm. Experiments have been carried out at several speed values of the conveyor belt, in the range from 5 to 180 mm/s.

Table 1 shows the correspondence between the belt speeds and the maximum time intervals from the visual detection of a part up to its effective grasping.

It can be observed that at the maximal speed of 180 mm/s, the robot-vision multitasking controller is still able to direct the SCARA-type manipulator to access visually detected, recognised and located objects.

Table 1 Correspondence between belt speed and part access time

Belt speed (mm/sec)	5	10	30	50	100	180
Grasping time (max) (sec)	1.4	1.6	1.9	2.0	2.3	2.5

In the experiment reported in Chap. 4.1, the vision library was successfully interfaced to an ABB 1570 vertical articulated robot.

The novelty of the research consist in developing an open architecture system for vision-based robot motion planning, allowing to use closed vision systems (here AdeptSight), that can be integrated with proprietary systems (for example AdeptSight has native functions which can be integrated only with Adept robots), with any other devices (robots, machines, feeders, a.o.) using standard communication mechanisms (serial line or Ethernet). Another novel contribution is the multitasking solution for picking objects in motion from any type of conveyor modelled as a $m \leq 3$ degree of freedom Cartesian robot.

References

1. Siciliano B, Sciavicco L, Villani L, Oriolo G (2010) Robotics, modelling, planning and control. Springer, Berlin
2. Borangiu Th, Ecaterina O, Manu M (2000) Multi-processor design of nonlinear robust motion control for rigid robots. Lecture notes in computer science, vol 1798. Springer, Berlin, pp 224–238
3. Borangiu Th (2002) Advanced robot motion control. Romanian Academy Press, Bucharest
4. Braun BM, Starr GP, Wood JE, Lumia R (2004) A framework for implementing cooperative motion on industrial controllers. IEEE Trans Robot Autom 20:583–589
5. Gueaieb W, Karray F, Al-Sharhan S (2003) A robust adaptive fuzzy position/force control scheme for cooperative manipulators. IEEE Trans Control Syst Technol 11:516–528
6. Battilotti S, Lanari L (1996) Tracking with disturbance attenuation for rigid robots. In: Proceedings of IEEE international conference on robot automation, Minneapolis, April 1996, pp 1570–1583
7. Borangiu Th, Anton F, Dumitrache A (2010) Robot programming. AGIR Publishing House, Bucharest
8. Kawasaki H, Ueki S, Ito S (2006) Decentralized adaptive coordinated control of multiple robot arms without using a force sensor. Automatica 42:481–488
9. Borangiu Th, Ionescu F, Manu M (2003) Visual servoing in robot motion control. In: Proceedings of 7th multi-conference on systemics, cybernetics and informatics SCI'03. Orlando, 27–30 July 2003, pp 987–992
10. Hutchinson S, Hager G, Corke P (1996) A tutorial on visual servo control. IEEE Trans Robot Autom 12:6561–6670
11. Xie WF, Li Z, Tu XW, Perron C (2009) Switching control of image based visual servoing with laser pointer in robotic assembly systems. IEEE Trans Ind Electron 520–529
12. Mendes JM, Restivo F, Leitao P, Colombo A (2010) Injecting service-orientation into multi-agent systems in industrial automation. Lecture notes in computer science, vol 6114, pp 313–320
13. Allotta B, Fioravanti D (2005) 3D motion planning for image-based visual servoing tasks. In: Proceedings of the IEEE international conference on robotics and automation, Barcelona, pp 2173–2178

14. Nelson BJ, Papanikoloupoulos P, Khosla PK (1996) Robotic visual servoing and robotic assembly tasks. IEEE Robot Autom Mag 23:97–102
15. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. J Comput Vis 60(2):91–110
16. Hossu A, Borangiu Th, Croicu A (1995) Robot Visionpro machine vision software for industrial training and applications. Version 2.2, Cat. #100062, Amsterdam, Tel Aviv, New Jersey, Eshed Robotec
17. Wilson W, Hulls C, Bell G (2006) Relative end-effector control using Cartesian position-based visual servoing. IEEE Trans Robot Autom 12:684–696
18. Miyabe T, Konno A, Uchiyama M, Yamano M (2004) An approach toward an automated object retrieval operation with a two-arm flexible manipulator. Int J Robot Res 23:275–291
19. ABB, Technical Reference Manual, RAPID Instructions, Functions, and Data types, 2004
20. Adept Reference Guide, V+ Programming, Adept Technology Inc. 2011
21. Bilen H, Hocaoglu M, Unel U, Sabanovic A (2012) Developing robust vision modules for microsystems applications. Mach Vis Appl 23(1):25–42
22. Borangiu Th, Ivanescu N, Brotac S (2002) An analytical method for visual robot -object calibration. In: Proceedings of the 7th international workshop robotics in Alpe-Adria-Danube region RAAD'98. Balatonfüred, pp 149–154
23. Chaumette F, Hutchinson S (2006) Visual servo control. Part I: basic approaches. IEEE Robot Autom Mag 13(4):82–90
24. Martinez-Rosas JC, Arteaga MA, Castillo-Sanchez A (2006) Decentralized control of cooperative robots without velocity-force measurements. Automatica 42:329–336
25. Gudiño-Lau J, Arteaga MA (2006) Dynamic model, control and simulation of cooperative robots: a case study, mobile robots, moving intelligence. ARS/pIV
26. Chaumette F, Hutchinson S (2005) A general and useful set of features for visual servoing. IEEE Trans Robot Autom 21:1116–1127
27. Lippiello V, Siciliano B, Villani L (2007) Position-based visual servoing in industrial multirobot cells using a hybrid camera configuration. IEEE Trans Robot 23:73–86
28. Borangiu Th (2004) Intelligent image processing in robotics and manufacturing. Romanian Academy Press, Bucharest
29. Corke P, Hutchinson S (2001) A new partitioned approach to image-based visual servo control. IEEE Trans Robot Autom 17:507–515
30. Lazar C, Burlacu A (2009) Visual servoing of robot manipulators using model-based predictive control. In: Proceedings of the 7th IEEE international conference on industrial informatics, Cardiff, pp 690–695
31. Lazar C, Burlacu A, Copot C (2011) Predictive control architecture for visual servoing of robot manipulators. In: Proceedings of the 18th IFAC world congress, Milano, pp 9464–9469