27

Visual Conveyor tracking in High-speed Robotics Tasks

Theodor Borangiu

1. Introduction

The chapter presents two methods and related motion control algorithms for robots which are required to pick "on-the-fly" objects randomly moving on conveyor belts; the instantaneous location of moving objects is computed by the vision system acquiring images from a stationary, down looking camera. The algorithms for visual tracking of conveyor belts for moving object access are 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 moving objects.

In the first method one assumes that conveyors are configured as external axes of the robot, which allows their modelling by means of a special class of variables called belt variables. A belt variable is here considered as a relative transformation (having a component variable in time) defining the location of a reference frame attached to the moving belt conveyor. By composing this time variable transformation (it reflects the contents of the belt's encoder) with the time – invariant instantaneous object location (estimated by vision for each object and compensated by the encoder offset value), the motion control algorithm will operate with a periodically updated destination, hence the robot will track the object moving on the belt.

In the second method the ensemble conveyor belt-actuator-sensor is configured as a $m \leq 3$-axis Cartesian robot, leading thus to a problem of cooperation between multiple robot manipulators subject to the multitasking control of a computer. Conceptually, the problem is solved by defining a number of *user tasks* which attach two types of "robots": the n – d.o.f. manipulator responsible with grasping on-the-fly objects moving on the conveyor belt, and the $m \leq 3$-axis robot emulating the conveyor belt under vision control. These user tasks run concurrently with the internal system tasks of a multitasking robot controller, mainly responsible for trajectory generation, axis servoing and system resources management.

Both methods use the concept of Conveyor Belt Window to implement fast reaction routines in response to emergency situations. The tracking algorithms

Open Access Database www.i-techonline.com

Source: Industrial-Robotics-Theory-Modelling-Control, ISBN 3-86611-285-8, pp. 964, ARS/plV, Germany, December 2006, Edited by: Sam Cubero

also provide collision-free object grasping by modelling the gripper's fingerprints and checking at run time whether their projections on the image plane cover only background pixels.

2. Modelling conveyors with belt variables

The problem of interest consists in building up a software environment allowing a robot controller to estimate the instantaneous position of the conveyor belt on which parts are travelling. The conveyor must be equipped with a displacement measuring device, in this case an *encoder*.
There are no constraints with respect to the position and orientation of the conveyor relative to the working area of the robot; the only requirement is that the belt's motion follows a straight line within the robot's manipulability region (Schilling, 1990). The encoder data will be interpreted by the controller as the current displacement of one of its external robot axes – in this case the tracked conveyor belt.

2.1 The special class of belt variables

The mechanism which allows specifying robot motions relative to a conveyor belt consists into modelling the belt by defining a special type of location data, named *belt variables*.

Definition 5.1: A *belt variable* is a relative homogenous transformation (having a component variable in time) which defines the location of a conveniently chosen reference frame attached to the conveyor's moving belt. The assignment of a belt variable is based on the software operation

DEFBELT %belt_variable = nominal_trans,scale_factor,

where:

- *%belt_variable* is the name of the belt variable to be defined, expressed as a 6-component homogenous transformation in minimal representation of the frame's orientation (e.g. by the Euler angles yaw, pitch and roll).
- *nominal_trans* represents the value in $\mathbb{R}^6$ of the relative transformation defining the position and the orientation of the conveyor belt. The X axis of *nominal_trans* indicates the direction of motion of the belt, the XY plane defined by this transformation is parallel to the conveyor's belt surface, and the position (X,Y,Z) specified by the transformation points to the approximate centre of the belt relative to the base frame of the robot. The origin of *nominal_trans* is chosen in the middle of the robot's working displacement over the conveyor.

- *scale_factor* is the calibrating constant specifying the ratio between the elementary displacement of the belt and one pulse of the encoder.

Using such a belt variable, it becomes possible to describe the relationship between a belt encoder and the location and speed of the reference frame (conveniently chosen with respect to the manipulability domain of the robot accessing the belt) which maintains a fixed position and orientation relative to the belt (Borangiu, 2004). The informational model of the conveyor belt is its assigned belt variable, to which additional modelling data must be specified for robot-vision compatibility:

- *window parameters*, defining the working area of the robot over the conveyor belt;
- *encoder offset*, used to adjust the origin of the belt's reference frame (e.g. relative to the moment when image acquisition is triggered).

The current orientation data in a belt variable is invariant in time, equal to that expressed by *nominal_trans*. In order to evaluate the current location updated by the same belt variable, the following real-time computation has to be performed: multiplication of a unit vector in the direction of $X_{|\mathrm{nominal_trans}}$ by *belt_distance* – a distance derived from the encoder's contents (periodically read by the system), and then addition of the result to the position vector of *nominal_trans*. The symbolic representation of this computation is given in equations (1) and (2):

$$XYZ_{\mathrm{instantaneous}} = XYZ_{\mathrm{nominal}} + belt_distance * \mathrm{unit_vect}(X_{|\mathrm{nominal_trans}}) \tag{1}$$

$$belt_distance = (encoder_count - encoder_offset) * scale_factor \tag{2}$$

Here, *encoder_count* is the encoder's read contents and *encoder_offset* will be used to establish the instantaneous location of the belt's reference frame (x_i, y_i) relative to its nominal location (x_n, y_n). In particular, the belt's offset can be used to nullify a displacement executed by the conveyor (by setting the value of the offset to the current value of the encoder's counter). The designed algorithm for visual robot tracking of the conveyor belt accounts for variable belt offsets which are frequently changed by software operations using mathematical expressions, like that included in the following V+ syntax: SETBELT %belt_variable = expression.

When the conveyor belt position is computed by referring to its assigned belt variable, the previously defined encoder offset will be always subtracted from the current position of the belt, i.e. from the encoder's current accumulated content. In the discussed method, setting a belt offset will use the real-valued

function BELT %belt_variable,mode to effectively reset the belt's position [encoder pulses].

Example 1:
A reaction routine continuously monitors a fast digital-input interrupt line which detects the occurrence of an external event of the type: "an object has completely entered the Conveyor Belt Window – and hence is completely visible". This event is detected by a photocell, and will determine an image acquisition of the moving object. The very moment the detected object is recognised as an object of interest and successfully located, the position of the belt is reset and consequently the belt variable will encapsulate from now on the current displacement of the belt relative to the belt's position in which the object has been successfully located. The V+ code is given below:

```
trigger = SIG(1001)          ;signal from photocell
save = PRIORITY  ;current priority of the robot-vision task
snap = 0                     ;reset event detection after image acquisition
success = 0           ;reset indication of successful part location
REACT –trigger,acquisition(snap,success,$name,belt_offset)
TIMER 1 = 0                  ;reset "timeout"-valued timer
IF TIMER(1)>timeout AND NOT snap  THEN
       GOTO l_end            ;no incoming parts, exit the task
 ELSE
LOCK PRIORITY + 2        ;raise priority to lock out any signals from the
          ;photocell until the current object is treated
IF success == 1  THEN
SETBELT %belt = belt_offset
IF $name == "PART"  THEN     ;if the object is of interest
Tracking the belt such that the robot picks on-the-fly the object (modelled
with the name "part") which was successfully located by vision in vis.loc
...
END
LOCK save ;re activate the REACT mechanism to check for on-off
;for on-off signal #1001 transitions
END
```

The interruption routine, automatically called by the REACT mechanism, has the form:

```
.PROGRAM acquisition(snap,success,$name,belt_offset)
 VPICTURE (cam) –1,1    ;image acquisition and recognition of one object
 snap = 1
;Locate any type of recognised object, return its name in the string
```

```
;var. $name and its location relative to the vision frame in vis_loc
 VLOCATE (cam,0) $name,vis.loc
 success = VFEATURE(1)          ;evaluate the success of the locating op.
 belt_offset = VFEATURE(8)
;The real-valued function VFEATURE(8) returns the contents of the
;belt's encoder when the strobe light for image acquisition was
;triggered.
 RETURN
.END
```

In what concerns the encoder's scale factor represented by the constant parameter *scale_factor*, it can be evaluated:

- either theoretically, knowing the mechanical coupling belt-encoder,
- or experimentally by reading the encoder's contents twice, each time when the image acquisition is triggered for a circular disk the presence of which is detected by the belt's photocell. The distance at which travel the two identical disks on the conveyor belt has been upstream set at a convenient, known value (see Fig. 1).

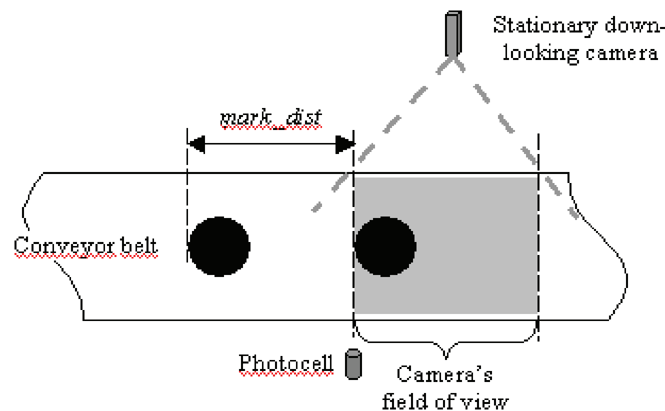

Figure 1. The experimental setup for conveyor belt calibration

2.2 The logical mechanism "Conveyor Belt Window" and emergency routines

There has been designed a logical mechanism called Conveyor Belt Window (CBW) which allows the user to check the area of the belt in which the robot will operate. A CBW defines a segment of the belt delimitated by two planes perpendicular to the direction of motion of the belt (this window is restricted only in the direction of motion of the conveyor's belt) (Borangiu & Kopacek, 2004).

Because the conveyor is modelled by a belt variable (e.g. *%belt_var*), in order to define a CBW it is necessary to refer the same belt variable and to specify two composed transformations which, together with the direction of motion of the belt, restrict the motion of the robot along a desired area of the conveyor:

WINDOW %belt_var = downstr_lim,upstr_lim,program_name,priority,

where:

- *downstr_lim* and *upstr_lim* are respectively the relative transformations defining the downstream and upstream edges of an invariant window positioned along the belt within the working space of the robot *and* the image field of the camera (it is necessary that the robot tracks and picks parts within these two limits);
- *program_name* indicates the reaction routine to be automatically called, whenever a window violation occurs while the robot tracks the conveyor belt;
- *priority* is the level of priority granted to the reaction routine. Normally, it must be greater than that of the conveyor tracking program, so that the motion of the robot can be immediately interrupted in case of window violation.

The CBW will be used not only in the stage of robot motion planning, but also at run time, during motion execution and tracking control, in order to check if the motion reference (the destination) is within the two imposed limits:

- When *a robot movement is planned*, the destination of the movement is checked against the operating CBW; if a window violation is detected, the reaction program is ignored and an error message will be issued.
- When *a robot movement relative to the conveyor belt is executed*, the destination is compared every 8 milliseconds with the window's limits; if a window violation is detected, the reaction program is automatically invoked according to its priority level and the robot will be instructed to stop tracking the belt.

There have been designed two useful CBW functions which allow the dynamic reconfiguring of programs, decisions, branching and loops during the execution of robot – vision conveyor tracking programs, function of the current value of the part-picking transformation relative to the belt, and of the current status of the belt tracking process. These functions are further introduced.
The function WINTEST(robot_transformation,time,mode) returns a value in millimetres indicating where is situated the location specified by the belt-relative composed transformation *robot_transformation*, with respect to the fixed window limits *downstr_lim* and *upstr_lim* at *time* seconds in the future, computed according to its current position and belt speed.

Finally, the argument *mode* is a real-valued expression which specifies whether the result of the WINTEST function represents a distance inside or outside the predefined conveyor belt window. For example, if *mode* is positive, the value returned by WINTEST will be interpreted as:

0: the composed, belt-relative location is inside the CBW;
<0: the location is upstream of *upstr_lim* of the CBW;
>0, the location is downstream of the *dwnstr_lim* of the CBW.

Hence, the returned value conforms to the WINDOW model shown in Fig. 2, for which the value returned by the function WINDOW increases as the belt-relative location moves downstream.

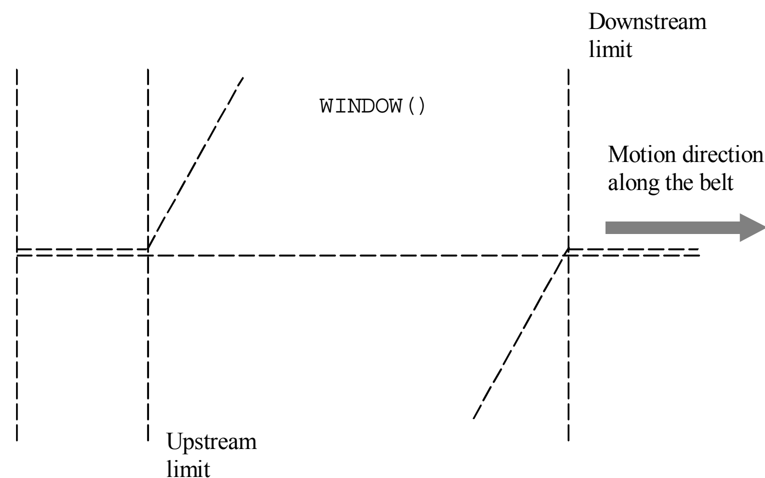

Figure 2. The WINDOW function for *mode* > 0

For robots tracking conveyor belts in order to pick moving objects recognised and located by vision, the belt-relative transformation is *%belt_var:part.loc* (variable in time), obtained by the composition of:

- *%belt_var*, models the conveyor belt,
- *part.loc*, is a composed, *time invariant* transformation expressing the gripper frame (x_g, y_g, z_g) relative to the base frame of the robot (x_0, y_0, z_0) at the moment the object was identified and located by vision.

For example, the distance WINTEST(%belt:part.loc,4,1) is positive if, in 4 seconds from the time being, the belt-relative part picking location will be outside the window defined for the conveyor belt modelled by *%belt*.
If the robot tries to move towards a belt-relative location that has not yet appeared inside the belt window (it is still upstream relative to the CBW), the motion control algorithm has been designed with two options:

- *temporarily stops the robot*, delaying thus the motion planning, until the time-variable destination enters the belt window;
- *definitively stops the robot* and generates immediately an error message.

Also, the control algorithm generates a condition of window violation anytime the vision-based robot motion planner computes a destination which is downstream the CBW, or will exit the CBW at the predicted time the robot will reach it. The function BELTSTATUS indicates the current status of the belt tracking process: *robot tracking the belt*; *destination upstream*; *destination downstream*; *window violation*, real-time information which can be used to dynamically reconfigure the robot – vision task.

2.3 Robot locations, frames and belt-relative movements planned by vision

To command the belt-relative motion of a robot with linear interpolation in the Cartesian space, i.e. to define an end-tip transformation relative to an instantaneous location of a moving frame (x_i, y_i) on the conveyor belt, the already defined belt variable (which models the conveyor belt as a relative transformation having time variable components along the *X* (and possibly *Y*) Cartesian axes) will be composed with the time-invariant end-tip transformation relative to the base of the robot (which is computed at run time by the vision part of the system).
The result will be a time-variable transformation updating the position reference for the robot. This reference or target destination tracks an object moving on the belt, to be picked by the robot's gripper. The target destination is:

- *planned once* at runtime by vision, as soon as the object is perfectly visible to the camera, either inside the manipulability area of the robot or upstream this area;

- *updated every 8 milliseconds* by the motion controller based on the current position data read from the belt's encoder, until the robot's end-point completes the necessary percentage of its motion segment towards the part's grasping location.

The research on Guidance Vision for Robots (GVR) accessing moving targets was directed to develop a *convergent* motion control algorithm for visually plan the motion of the robot as a result of object detection, recognition and locating on a moving conveyor belt, and than track the object in order to grasp it inside a conveniently defined belt window. The main idea relies on dynamically changing the visually computed destination of the robot end point by composing it with a belt-related transformation updated every 8 milliseconds from the encoder data.

If a stationary camera looks down at the conveyor belt, and supposing that its field of view covers completely a conveyor belt window defined inside the working area of the robot (after execution of a camera – robot calibration session), then the image plane can be referred by the time – invariant frame (x_{vis}, y_{vis}) as represented in Fig. 3.

It is also assumed that the X axes of the reference frame of the robot (x_0), of the conveyor's direction of motion (x_n) and of the image plane (x_{vis}) are parallel. The conveyor belt is modelled by the belt variable %belt. Parts are circulating on the belt randomly; their *succession* (current part type entering the CBW), *distance from the central axis of the conveyor* and *orientation* are unknown. The "Look-and-Move" interlaced operating principle of the image processing section and motion control section is used (Hutchinson, 1996), (Borangiu, 2001), (West, 2001), (Adept, 2001). According to this principle, while an image of the CBW is acquired and processed for object identification and locating, no motion command is issued and reciprocally, the camera will not snap images while the robot tracks a previously located part in order to pick it "on-the-fly".

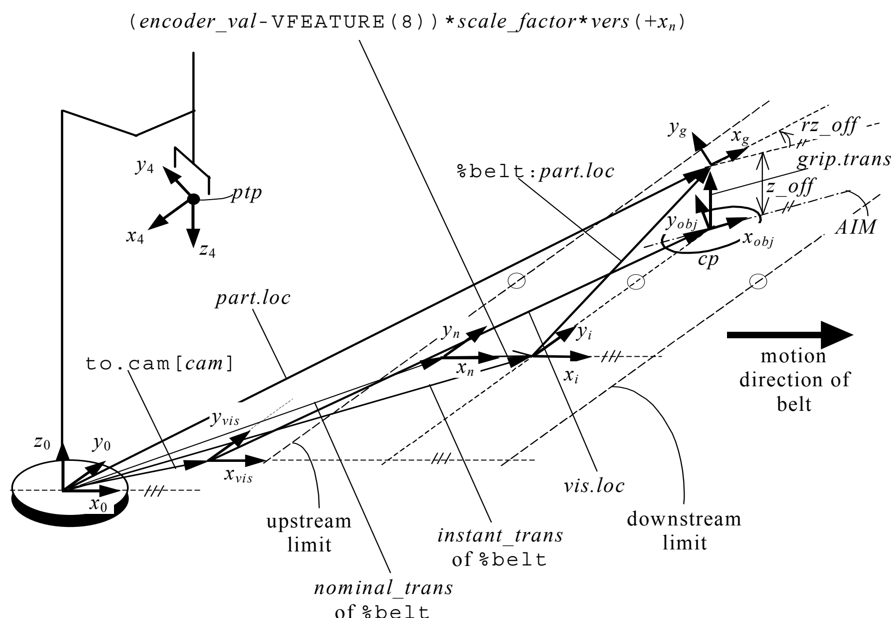

Figure 3. Robot-Vision and belt-relative transformations for conveyor tracking

The robot motion control algorithm for tracking the conveyor belt in order to pick "on-the-fly" one by one objects recognised and located by vision computing consists of the following basic steps:

1. *Triggering the strobe light* (synch./asynch. relative to the read cycle of the video camera) when *image acquisition* is requested from a fast digital-input interrupt line connected to a photocell mounted at the upstream limit of the CBW. The interrupt line signals that an object has completely entered the belt window.
2. *Recognizing a single object* that just completely entered the belt window. Object recognition is exclusively based in this approach on the match with previously learned models of all objects of interest (Lindenbaum, 1997).
3. *Locating the object* which was recognised, by computing the coordinates of its centre of mass and the angle between its minimum inertia axis (MIA) and x_{vis}. As can be seen in Fig. 3, the object-attached frame (x_{obj}, y_{obj}) has the abscissa aligned with the minimum inertia axis (MIA), and the current location of the object in the image plane is computed by the vision section and returned in *vis.loc*.
4. *Planning the instantaneous destination of the robot*. Once the object is *recognized* as the instance of a model and *located*, the related grasping transformation *grip.trans* is called. Assuming that the grasping style is such that the projection of the gripper's centre on the image plane coincides with the object's centre of mass, the gripper-attached frame (x_g, y_g) will be offset relative to the object-attached frame along z_0 by *z_off* millimetres and turned with *r_off* degrees about z_0. Now, using the relative transformation to.cam[*cam*](as output of the camera-robot calibration session) relating the vision frame $(x_{vis}, y_{vis}, z_{vis})$ to the base frame of the robot (x_0, y_0, z_0), the current destination of the robot (for a frozen conveyor belt) is computed from the vision data as a composed transformation *part.loc*, expressing the gripper frame relative to the robot base frame:

   part.loc = to.cam[cam]:vis.loc:grip.trans

5. *Synchronising the encoder belt with the motion of the object* recognized in the belt window. This operation consists into setting the offset of the conveyor belt at a correct value. The operation

   SETBELT %belt = encoder_val(strobe)

establishes the point of interest of the conveyor belt modelled with %belt as the point corresponding to the current value *encoder_val*(*strobe*) of the encoder counter at the time the strobe light was triggered. This value is available immediately after object locating. Thus, as soon as one object is recognized and located, the current belt position, identified by (x_i, y_i), will be reset since:

$$\begin{aligned} xyz_i = xyz_n + (encoder_val - encoder_val(strobe)) * \\ * scale_factor * \text{unit_vect}(x_n) = xyz_n \end{aligned} \tag{3}$$

6. *Tracking and picking the object moving on the belt*. This requires issuing a linear motion command in the Cartesian space, relative to the belt. A composed relative transformation %belt:part.loc, expressing the current computed location of the gripper relative to the instantaneous moving frame (x_i, y_i), is defined. Practically, the tracking procedure begins immediately after the instantaneous position of the belt – expressed by the frame (x_i, y_i) has been initialized by the SETBELT operation, and consists into periodically updating the destination of the gripper by shifting it along the x_n axis with encoder counts accumulated during successive sampling periods $\ldots, t_{k-1}, t_k, t_{k+1}, \ldots \quad \Delta t = t_{k+1} - t_k = \text{const}:$

$$\begin{aligned} \%belt : part.loc|_{t_{k+1}} = \texttt{SHIFT}\,(\%belt : part.loc|_{t_k}\ \texttt{BY}\ldots \\ \ldots\ encoder_count(t_{k+1} - t_k) * scale_factor, 0, 0) \end{aligned} \tag{4}$$

```
MOVES %belt:part.loc        ;go towards the moving target
CLOSEI                      ;grasp "on the fly" the object
```

7. Once the robot commanded towards a destination relative to the belt, the gripper will continuously track the belt until a new command will be issued to approach a location which is not relative to the belt.

For belt-relative motions, the destination changes continuously; depending on the magnitude and the variations of the conveyor speed it is possible that the robot will not be able to attain the final positions within the default error tolerance.

In such cases, the error tolerance must be augmented. In extreme cases it will be even necessary to totally deactivate the test of final error tolerance. Fig. 4

presents the designed robot motion control algorithm for tracking the conveyor belt in order to pick "on-the-fly" an object recognized and located by vision computation inside the a priori defined belt window. A REACT mechanism continuously monitors the fast digital-input interrupt line which signals that an object has completely entered the belt window. The robot motion relative to the belt will be terminated:

- when moving the robot towards a *non belt-relative location* or
- when a *window violation* occurs.

***Example 2*:**
The following series of instructions will move the robot end-effector towards a belt-relative location part_2 (the belt is modelled as %belt[1]), open the gripper, track the conveyor belt for 5 seconds (in fact the location part_2 on the belt), close the gripper and finally leave the belt to move towards a fixed location.

```
MOVES %belt[1]:part_2
OPENI
DELAY 5.0
CLOSEI
MOVES fixed_location
```

When defining the Conveyor Belt Window, a special high-priority routine can be specified, which will be automatically invoked to correct any window violation during the process of tracking moving objects. In such situations the robot will be accelerated (if possible) and the downstream limit temporarily shifted in the direction of motion of the conveyor belt (within a timeout depending on the belt's speed) in which the error tolerance must be reached (Espiau, 1992), (Borangiu, 2002).

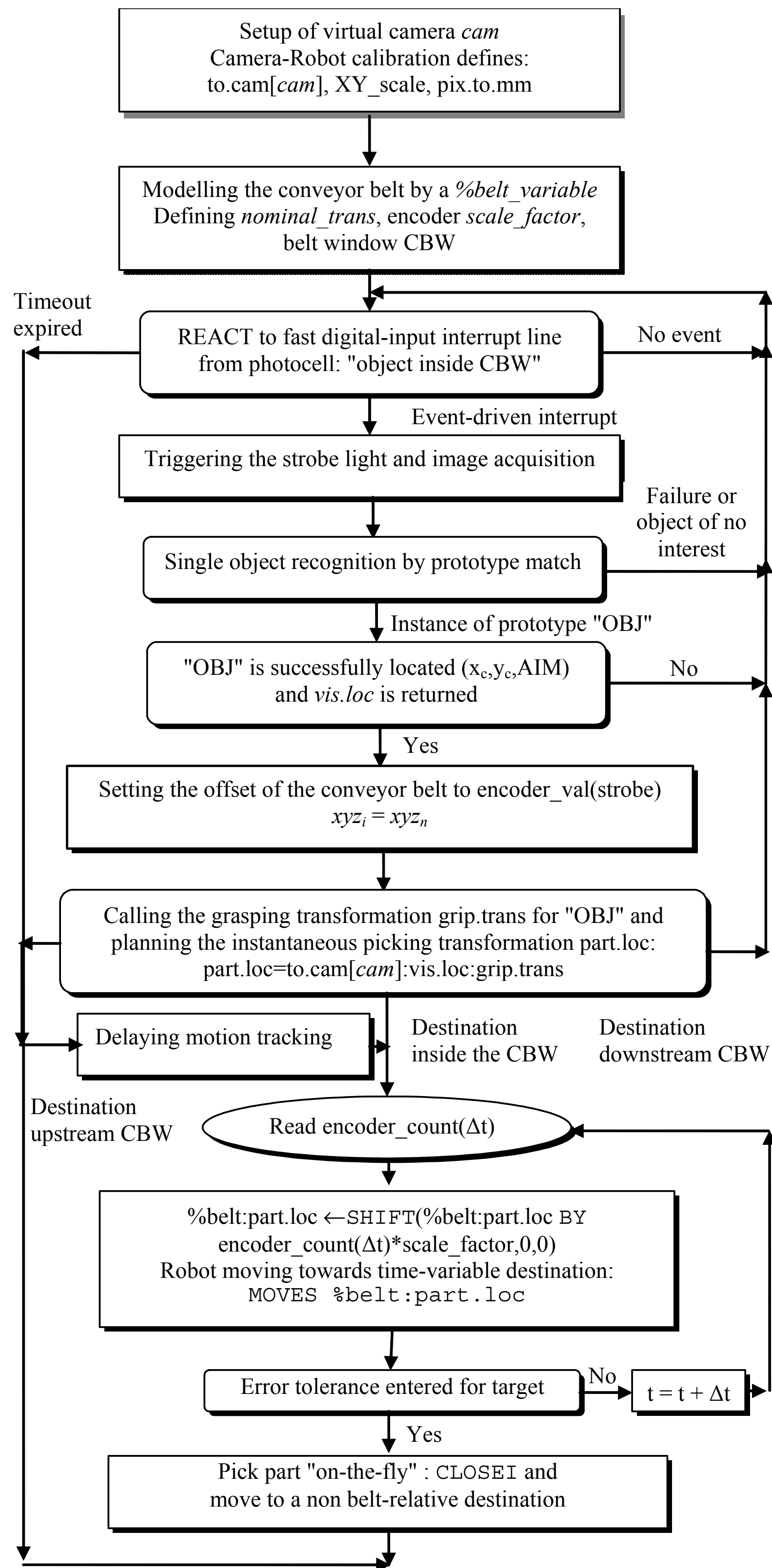

Figure 4. The robot motion algorithm for visual tracking of the conveyor belt

3. Tracking conveyors as $m \leq 3$ Cartesian axis robots

According to the second tracking method, the ensemble conveyor belt + actuator + sensor is configured as an $m \leq 3$*-axis Cartesian robot*, which leads to a problem of cooperation between multiple robots subject to multitasking computer control. The V+ structured programming environment is used for exemplifying the multi tasking control of robots visually tracking moving objects on multiple belts.

3.1 Multitasking control for robot cooperation

Conceptually, the problem is solved by defining a number of *user tasks* which attach two types of "robots": the n – d.o.f. manipulator responsible with grasping on-the-fly objects moving on the conveyor belt, and the $m \leq 3$-axis robot emulating the conveyor belt under vision control. These user tasks run concurrently with the internal system tasks of a multitasking robot controller, mainly responsible for trajectory generation, axis servoing and system resource management (Adept, 2001).

In this respect, there are three tasks to be at least defined for the tracking problem:

1. Task 1: Dynamic re-planning the destination location (grasping the moving object) for the robot manipulator.
2. Task 2: Continuously moving (driving) the $m \leq 3$-axis vision belt. In the most general case, the belt moves along any 3D-direction relative to the robot base frame (x_0, y_0, z_0).
3. 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.

3.1.1 Specifying tasks, time slices and priorities

A multitasking robot control system appears to execute all these program tasks at the same time. However, this is actually achieved by rapidly switching between the tasks many times each second, each task receiving a fraction of the total time available. This is referred to as concurrent execution (Zhuang, 1992), (Borangiu, 2005).

The amount of time a particular program task receives is caused by two parameters: its *assignment* to the various time slices, and its *priority* within the time slice. One assumes that, in the multitasking operating system, each *system cycle* is divided into 16 *time slices* of one millisecond each, the slices being numbered 0 through 15. A single occurrence of all 16 time slices is referred to

as a *major cycle*. For a robot each of these cycles corresponds to one output from the trajectory generator to the servos.
A number of seven user tasks, e.g. from 0 to 6, will be used and their configuration tailored to suit the needs of specific applications. Each program task configured for use requires dedicated memory, which is not available to user programs. Therefore, the number of tasks available should be made no larger than necessary, especially if memory space for user programs is critical.
When application programs are executed, their program tasks are normally assigned default time slices and priorities according to the current system configuration. The defaults can be overridden temporarily for any user program task, by specifying the desired time slice and priority parameters of the EXECUTE initiating command.
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	Do not run in this slice even if no other task is ready to run.
0	Do not run in this slice unless no other task from this slice is ready to run.
1-64	Run in this slice according to specified priority. Higher priority tasks may lock lower ones. Priorities are broken into the following ranges:
1-31	Normal user task priorities;
32-62	Used by robot controller's *device drivers* and *system tasks*;
63	Used by *trajectory generator*. Do not use 63 unless you have very short task execution times, because use of these priorities may cause jerks in the robot trajectories;
64	Used by the *servo*. Do not use 64 unless you have very short task execution times, because use of these priorities may cause jerks in the robot trajectories.

The V+ operating system has a number of *internal* (system) *tasks* that compete with *application* (user) *program tasks* for time within each time slice:

- On motion systems, the V+ *trajectory generator* runs (at the highest priority task) in slice #0 and continues through as many time slices as necessary to compute the next motion device set point.

- On motion systems, the CPU running servo code runs the *servo task* (at interrupt level) every 1 or 2 milliseconds (according to the controller configuration utility).

The remaining time is allocated to user tasks, by using the controller configuration utility. For each time slice, you specify which tasks may run in the slice and what priority each task has in that slice.

3.1.2 Scheduling of program execution 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 or 1 in "Look-and-Move" interlaced robot motion control, due to the continuous, high priority assignment of these two tasks, over the first 13 time slices. However, vision guidance and motion re-planning programs complete their computation in less than the 13 time slices (0-12).
Consequently, 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.
A WAIT *condition* instruction with no argument will suspend then, once per loop execution, the motion re-planning program, executing on user task 1, until the start of the next major cycle (slice 0). At that time, the "vision processing and belt tracking" task becomes runnable and will execute, due to its high priority assignment.
Due to their reduced amount of computation, programs related to the management of the conveyor belt should be always assigned to tasks 2, 3, 5 or 6 if the default priority scheme is maintained for user program tasks, leaving tasks 1 and 2 for the intensive computational vision and robot motion control.
Whenever the current task becomes inactive, the multitasking OS searches for a new task to run. The search begins with the highest priority task in the current time slice and proceeds through in order of descending priority. If multiple programs wait to run in the task, they are run according to relative program priorities. If a runnable task is not found, the next higher slice is checked. All time slices are checked, wrapping around from slice 15 to slice 0 until the original slice is reached. Whenever a 1 ms interval expires, the system performs a similar search of the next time slice; if this one does not contain a runnable task, the currently executing 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*. Whenever a member of a round-robin group is selected by the normal slice searching, the group is scanned to find the member of the group that run most recently. The member that follows the most recent is run instead of the one which was originally selected.

The V+ RELEASE program instruction may be used to bypass the normal scheduling process by explicitly passing control to another task. That task then goes to run in the current time slice until it is rescheduled by the 1 ms clock. A task may also RELEASE to *anyone*, which means that a normal scan is made of all other tasks to find one that is ready to run. During this scan, members of the original task's round-robin group (if any) are ignored. Therefore, a RELEASE to anyone cannot be used to pass control to a different member of the current group.
Round-robin groups are treated as a single task. If any member of the group is selected during the scan, then the group is selected. The group is scanned to find the task in the group following the one which ran most recently, and that task is run. Within each time slice, the task with highest priority can be locked out only by a servo interrupt. Tasks with lower priority, defined for driving the conveyor belt and reading position data from its encoder, can run only if the higher-priority task, defined for vision guidance of the *n*–d.o.f. robot and for tracking the 1–d.o.f. robot-like conveyor belt, is inactive or waiting. A user task waits whenever:

- The program issues an input or an output request that causes a wait.
- The program executes a robot motion instruction while the robot is still moving in response to a previous motion instruction.
- The program executes a WAIT or WAIT.EVENT program instruction.

If a program is executing continuously without performing any of the above operations, it locks out any lower-priority tasks in its time slice. Thus, programs that execute in continuous loops, like vision guidance and motion replanning for belt tracking, should generally execute a WAIT (or WAIT.EVENT) instruction occasionally (for example, *once each time through the loop*).
If a program potentially has a lot of *critical processing* to perform, its task should be in *multiple slices*, and the task should have the *highest priority* in these slices. This will guarantee the task's getting all the time needed in the multiple slices, plus (if needed) additional unused time in the major cycle.

Fig. 5 shows the *task scheduler algorithm* which was designed for an *n*-d.o.f. robot tracking a continuously updated object grasping location, and picking the object "on-the-fly" from a conveyor belt, when motion completes. The object is recognized and located by vision, and updating of its position is provided by encoder data reads from the conveyor belt modelled as a 1-d.o.f. robot. The priority analysis and round-robin member selection are also indicated.
The problem of conveyor tracking with vision guiding for part identification and locating required definition of three user tasks, to which programs were associated:

Thank You for previewing this eBook

You can read the full version of this eBook in different formats:

- HTML (Free /Available to everyone)
- PDF / TXT (Available to V.I.P. members. Free Standard members can access up to 5 PDF/TXT eBooks per month each month)
- Epub & Mobipocket (Exclusive to V.I.P. members)

To download this full book, simply select the format you desire below