4

Advanced Techniques of Industrial Robot Programming

Frank Shaopeng Cheng
Central Michigan University
United States

1. Introduction

Industrial robots are reprogrammable, multifunctional manipulators designed to move parts, materials, and devices through computer controlled motions. A robot application program is a set of instructions that cause the robot system to move the robot's end-of-arm-tooling (or end-effector) to robot points for performing the desired robot tasks. Creating accurate robot points for an industrial robot application is an important programming task. It requires a robot programmer to have the knowledge of the robot's reference frames, positions, software operations, and the actual programming language. In the conventional "lead-through" method, the robot programmer uses the robot teach pendant to position the robot joints and end-effector via the actual workpiece and record the satisfied robot pose as a robot point. Although the programmer's visual observations can make the taught robot points accurate, the required teaching task has to be conducted with the real robot online and the taught points can be inaccurate if the positions of the robot's end-effector and workpiece are slightly changed in the robot operations. Other approaches have been utilized to reduce or eliminate these limitations associated with the online robot programming. This includes generating or recovering robot points through user-defined robot frames, external measuring systems, and robot simulation software (Cheng, 2003; Connolly, 2006; Pulkkinen1 et al., 2008; Zhang et al., 2006).

Position variations of the robot's end-effector and workpiece in the robot operations are usually the reason for inaccuracy of the robot points in a robot application program. To avoid re-teaching all the robot points, the robot programmer needs to identify these position variations and modify the robot points accordingly. The commonly applied techniques include setting up the robot frames and measuring their positional offsets through the robot system, an external robot calibration system (Cheng, 2007), or an integrated robot vision system (Cheng, 2009; Connolly, 2007). However, the applications of these measuring and programming techniques require the robot programmer to conduct the integrated design tasks that involve setting up the functions and collecting the measurements in the measuring systems. Misunderstanding these concepts or overlooking these steps in the design technique will cause the task of modifying the robot points to be ineffective.

Robot production downtime is another concern with online robot programming. Today's robot simulation software provides the robot programmer with the functions of creating

virtual robot points and programming virtual robot motions in an interactive and virtual 3D design environment (Cheng, 2003; Connolly, 2006). By the time a robot simulation design is completed, the simulation robot program is able to move the virtual robot and end-effector to all desired virtual robot points for performing the specified operations to the virtual workpiece without collisions in the simulated workcell. However, because of the inevitable dimensional differences of the components between the real robot workcell and the simulated robot workcell, the virtual robot points created in the simulated workcell must be adjusted relative to the actual position of the components in the real robot workcell before they can be downloaded to the real robot system. This task involves the techniques of calibrating the position coordinates of the simulation Device models with respect to the user-defined real robot points.

In this chapter, advanced techniques used in creating industrial robot points are discussed with the applications of the FANUC robot system, Delmia IGRIP robot simulation software, and Dynalog DynaCal robot calibration system. In Section 2, the operation and programming of an industrial robot system are described. This includes the concepts of robot's frames, positions, kinematics, motion segments, and motion instructions. The procedures for teaching robot frames and robot points online with the real robot system are introduced. Programming techniques for maintaining the accuracy of the exiting robot points are also discussed. Section 3 introduces the setup and integration of a two dimensional (2D) vision system for performing vision-guided robot operations. This includes establishing integrated measuring functions in both robot and vision systems and modifying existing robot points through vision measurements for vision-identified workpieces. Section 4 discusses the robot simulation and offline programming techniques. This includes the concepts and procedures related to creating virtual robot points and enhancing their accuracy for a real robot system. Section 5 explores the techniques for transferring industrial robot points between two identical robot systems and the methods for enhancing the accuracy of the transferred robot points through robot system calibration. A summary is then presented in Section 6.

2. Creating Robot Points Online with Robot

The static positions of an industrial robot are represented by Cartesian reference frames and frame transformations. Among them, the robot base frame R(x, y, z) is a fixed one and the robot's default tool-center-point frame Def_TCP (n, o, a), located at the robot's wrist faceplate, is a moving one. The position of frame Def_TCP relative to frame R is defined as the robot point $P[n]^{R}_{Def\_TCP}$ and is mathematically determined by the 4 × 4 homogeneous transformation matrix in Eq. (1)

$$P[n]^{R}_{Def\_TCP} = {}^{R}T_{Def\_TCP} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1)$$

where the coordinates of vector $p = (p_x, p_y, p_z)$ represent the location of frame Def_TCP and the coordinates of three unit directional vectors n, o, and a represent the orientation of frame

Def_TCP. The inverse of ${}^{R}T_{Def\_TCP}$ or $P[n]^{R}_{Def\_TCP}$ denoted as $({}^{R}T_{Def\_TCP})^{-1}$ or $(P[n]^{R}_{Def\_TCP})^{-1}$ represents the position of frame R to frame Def_TCP, which is equal to frame transformation ${}^{Def\_TCP}T_{R}$. Generally, the definition of a frame transformation matrix or its inverse described above can be applied for measuring the relative position between any two frames in the robot system (Niku, 2001). The orientation coordinates of frame Def_TCP in Eq. (1) can be determined by Eq. (2)

$$\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = \mathrm{Rot}(z,\theta_z)\mathrm{Rot}(y,\theta_y)\mathrm{Rot}(x,\theta_x) \quad , (2)$$

$$= \begin{bmatrix} \cos\theta_z\cos\theta_y & \cos\theta_z\sin\theta_y\sin\theta_x - \sin\theta_z\cos\theta_x & \cos\theta_z\sin\theta_y\cos\theta_x + \sin\theta_z\sin\theta_x \\ \sin\theta_z\cos\theta_y & \sin\theta_z\sin\theta_y\sin\theta_x + \cos\theta_z\cos\theta_x & \sin\theta_z\sin\theta_y\cos\theta_x - \cos\theta_z\sin\theta_x \\ -\sin\theta_y & \cos\theta_y\sin\theta_x & \cos\theta_y\cos\theta_x \end{bmatrix}$$

where transformations Rot(x, θ_x), Rot(y, θ_y), and Rot(z, θ_z) are pure rotations of frame Def_TCP about the x-, y-, and z-axes of frame R with the angles of θ_x (yaw), θ_y (pitch), and θ_z (roll), respectively. Thus, a robot point $P[n]^{R}_{Def\_TCP}$ can also be represented by Cartesian coordinates in Eq. (3)

$$P[n]^{R}_{Def\_TCP} = (x, y, z, w, p, r) \cdot \quad (3)$$

It is obvious that the robot's joint movements are to change the position of frame Def_TCP. For an n-joint robot, the geometric motion relationship between the Cartesian coordinates of a robot point $P[n]^{R}_{Def\_TCP}$ in frame R (i.e. the robot world space) and the proper displacements of its joint variables q = (q_1, q_2, ..q_n) in robot joint frames (i.e. the robot joint space) is mathematically modeled as the robot's kinematics equations in Eq. (4)

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_{11}(q,r) & f_{12}(q,r) & f_{13}(q,r) & f_{14}(q,r) \\ f_{21}(q,r) & f_{22}(q,r) & f_{23}(q,r) & f_{24}(q,r) \\ f_{31}(q,r) & f_{32}(q,r) & f_{33}(q,r) & f_{34}(q,r) \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4)$$

where f_{ij}(q, r) (for i = 1, 2, 3 and j = 1, 2, 3, 4) is a function of joint variables q and joint parametersr.

Specifically, the robot forward kinematics equations will enable the robot system to determine where a $P[n]^{R}_{Def\_TCP}$ will be if the displacements of all joint variables q=(q_1, q_2, ..q_n) are known. The robot inverse kinematics equations will enable the robot system to calculate what displacement of each joint variable q_k (for k = 1 ,..., n) must be if a $P[n]^{R}_{Def\_TCP}$ is specified. If the inverse kinematics solutions for a given $P[n]^{R}_{Def\_TCP}$ are infinite, the robot system defines the point as a robot "singularity" and cannot move frame Def_TCP to it.

In robot programming, the robot programmer creates a robot point $P[n]^{R}_{Def\_TCP}$ by first declaring it in a robot program and then defining its coordinates in the robot system. The conventional method is through recording a particular robot pose with the robot teach pendent (Rehg, 2003). Under the teaching mode, the robot programmer jogs the robot's joints for poisoning the robot's end-effector relative to the workpiece. As joint k moves, the serial pulse coder of the joint measures the joint displacement q_k relative to the "zero" position of the joint frame. The robot system substitutes all measured values of $q = (q_1, q_2, ..q_n)$ into the robot forward kinematics equations to determine the corresponding Cartesian coordinates of frame Def_TCP in Eq. (1) and Eq. (3). After the robot programmer records a $P[n]^{R}_{Def\_TCP}$ with the teach pendant, its Cartesian coordinates and the corresponding joint values are saved in the robot system. The robot programmer may use the "Representation" softkey on the teach pendant to automatically convert and display the joint values and Cartesian coordinates of a taught robot point $P[n]^{R}_{Def\_TCP}$. It is important to notice that Cartesian coordinates in Eq. (3) is the standard representation of a $P[n]^{R}_{Def\_TCP}$ in the industrial robot system, and its joint representation always uniquely defines the position of frame Def_TCP (i.e. the robot pose) in frame R.

In robot programming, the robot programmer defines a motion segment of frame Def_TCP by using two taught robot points in a robot motion instruction. During the execution of a motion instruction, the robot system utilizes the trajectory planning method called "linear segment with parabolic blends" to control the joint motion and implement the actual trajectory of frame Def_TCP through one of the two user-specified motion types. The "joint" motion type allows the robot system to start and end the motion of all robot joints at the same time resulting in an unpredictable, but repeatable trajectory for frame Def_TCP. The "Cartesian" motion type allows the robot system to move frame Def_TCP along a user-specified Cartesian path such as a straight line or a circular arc in frame R during the motion segment, which is implemented in three steps. First, the robot system interpolates a number of intermediate points along the specified Cartesian path in the motion segment. Then, the proper joint values for each interpolated robot point are calculated by the robot inverse kinematics equations. Finally, the "joint" motion type is applied to move the robot joints between two consecutive interpolated robot points.

Different robot languages provide the robot systems with motion instructions in different format. The motion instruction of FANUC Teach Pendant Programming (TPP) language (Fanuc, 2007) allows the robot programmer to define a motion segment in one statement that includes the robot point P[n], motion type, speed, motion termination type, and associated motion options. Table 1 shows two motion instructions used in a FANUC TP program.

FANUC TPP Instruction	Description
1. J P[1] 50% FINE	Moves the TCP frame to robot point P[1] with "Joint" motion type (J) and at 50% of the default joint maximum speed, and stops exactly at P[1] with a "Fine" motion termination.
2. L P[2] 100 mm/sec FINE	Utilizes "Linear" motion type (L) to move TCP frame along a straight line from P[1] to P[2] with a TCP speed of 100 mm/sec and a "Fine" motion termination type.

Table 1. Motion instructions of FANUC TPP language

2.1 Design of Robot User Tool Frame

In the industrial robot system, the robot programmer can define a robot user tool frame UT[k](x, y, z) relative to frame Def_TCP for representing the actual tool-tip point of the robot's end-effector. Usually, the UT[k] origin represents the tool-tip point and the z-axis represents the tool axis. A UT[k] plays an important role in robot programming as it not only defines the actual tool-tip point but also addresses its variations. Thus, every end-effector used in a robot application must be defined as a UT[k] and saved in robot system variable UTOOL[k]. Practically, the robot programmer may directly define and select a UT[k] within a robot program or from the robot teach pendant. Table 2 shows the UT[k] frame selection instructions of FANUC TPP language. When the coordinates of a UT[k] is set to zero, it represents frame Def_TCP. The robot system uses the current active UT[k] to record a robot point $P[n]^{R}_{UT[k]}$ as shown in Eq. (5) and cannot move the robot to any robot point $P[m]^{R}_{UT[g]}$ that is taught with a UT[g] different from UT[k] (i.e. $g \neq k$).

$$P[n]^{R}_{UT[k]} = {}^{R}T_{UT[k]} \tag{5}$$

It is obvious that a robot point $P[n]^{R}_{Def\_TCP}$ in Eq. (1) or Eq. (3) can be taught with different UT[k], thus, represented in different Cartesian coordinates in the robot system as shown in Eq. (6)

$$P[n]^{R}_{UT[k]} = P[n]^{R}_{Def\_TCP} \times {}^{Def\_TCP}T_{UT[k]} \ . \tag{6}$$

FANUC TPP Instruction	Description
1. UTOOL_NUM=1	Set UT[1] frame to be the current active UT.

Table 2. UT[k] frame selection instructions of FANUC TPP language

To define a UT[k] for an actual tool-tip point $P_{T\text{-}Ref}$ whose coordinates (x, y, z, w, p, r) in frame D_{ef_TCP} is unknown, the robot programmer must follow the UT Frame Setup procedure provided by the robot system and teach six robot points $P[n]^{R}_{Def\_TCP}$ (for n = 1, 2, … 6) with respect to $P_{T\text{-}Ref}$ and a reference point $P_{S\text{-}Ref}$ on a tool reachable surface. The "three-point" method as shown in Eq. (7) and Eq. (8) utilizes the first three taught robot points in the UT Frame Setup procedure to determine the UT[k] origin. Suppose that the coordinates of vector ${}^{Def\_TCP}p = [p_n, p_o, p_a]^T$ represent point $P_{T\text{-}Ref}$ in frame Def_TCP. Then, it can be determined in Eq. (7)

$${}^{Def\_TCP}p = (T_1)^{-1} \times {}^{R}p, \tag{7}$$

where the coordinates of vector ${}^{R}p = [p_x, p_y, p_z]^T$ represents point $P_{T\text{-}Ref}$ in frame R and T_1 represents the first taught robot point $P[1]^{R}_{Def\_TCP}$ when point $P_{T\text{-}Ref}$ touches point $P_{S\text{-}Ref}$. The coordinates of vector ${}^{R}p = [p_x, p_y, p_z]^T$ also represents point $P_{S\text{-}Ref}$ in frame R and can be solved by the three linear equations in Eq. (8)

$$(\mathrm{I} - \mathrm{T}_2\mathrm{T}_3^{-1}) \times {}^{\mathrm{R}}\mathrm{p} = 0 \ , \tag{8}$$

where transformations T_2 and T_3 represent the other two taught robot points $P[2]^{R}_{Def\_TCP}$ and $P[3]^{R}_{Def\_TCP}$ in the UT Frame Setup procedure respectively when point $P_{T\text{-}Ref}$ is at point $P_{S\text{-}Ref}$. To ensure the UT[k] accuracy, these three robot points must be taught with point $P_{T\text{-}Ref}$ touching point $P_{S\text{-}Ref}$ from three different approach statuses. Practically, $P[2]^{R}_{Def\_TCP}$ (or $P[3]^{R}_{Def\_TCP}$) can be taught by first rotating frame Def_TCP about its x-axis (or y-axis) for at least 90 degrees (or 60 degrees) when the tool is at $P[1]^{R}_{Def\_TCP}$, and then moving point $P_{T\text{-}Ref}$ back to point $P_{S\text{-}Ref}$. A UT[k] taught with the "three-point" method has the same orientation of frame Def_TCP.

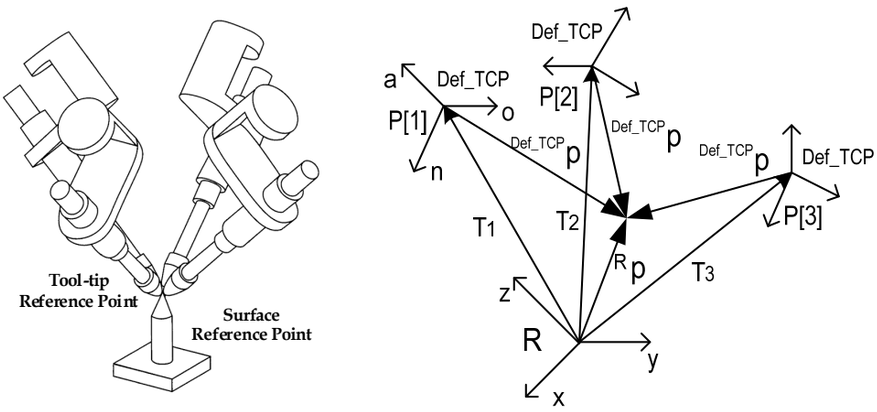


Fig. 1. The three-point method in teaching a UT[k]

If the UT[k] orientation needs to be defined differently from frame Def_TCP, the robot programmer must use the "six-point" method and teach additional three robot points required in UT Frame Setup procedure. These three points define the orient origin point, the positive x-direction, and the positive z-direction of the UT[k], respectively. The method of using such three non-collinear robot points for determining the orientation of a robot frame is to be discussed in section 2.2.

Due to the tool change or damage in robot operations the actual tool-tip point of a robot's end-effector can be varied from its taught UT[k], which causes the inaccuracy of existing robot points relative to the workpiece. To aviod re-teaching all robot points, the robot programmer needs to teach a new UT[k]' for the changed tool-tip point and shift all existing robot points through offset ${}^{Def\_TCP}T_{Def\_TCP'}$ as shown in Fig. 2. Assume that transformation ${}^{Def\_TCP}T_{UT[k]}$ represents the position of the original tool-tip point and remains unchanged when frame UT[k] changes into new UT[k]' as shown in Eq. (9)

$${}^{Def\_TCP}T_{UT[k]} = {}^{Def\_TCP'}T_{UT[k]'}, \tag{9}$$

where frame Def_TCP' represents the position of frame Def_TCP after frame UT[k] moves

to UT[k]'. In this case, the pre-taught robot point $P[n]^{R}_{UT[k]}$ can be shifted into the corresponding robot point $P[n]^{R}_{UT[k]'}$ through Eq. (10)

$$^{Def\_TCP}T_{UT[k]'} = {}^{Def\_TCP}T_{Def\_TCP'} \times {}^{Def\_TCP'}T_{UT[k]'}. \quad (10)$$

The industrial robot system usually implements Eq. (9) and Eq. (10) as both a system utility function and a program instruction. As a system utility function, the offset $^{Def\_TCP}T_{Def\_TCP'}$ changes the position of frame Def_TCP in the robot system so that the robot programmer is able to change the current UT[k] of a taught P[n] into a different UT[k]' while remaining the same Cartesian coordinates of P[n] in frame R. As a program instruction, $^{Def\_TCP}T_{Def\_TCP'}$ shifts the pre-taught robot point $P[n]^{R}_{UT[k]}$ into the corresponding point $P[n]'^{R}_{UT[k]'}$ without changing the position of frame Def_TCP. Table 3 shows the UT[k] offset instruction of FANUC TPP language for Eq. (10).

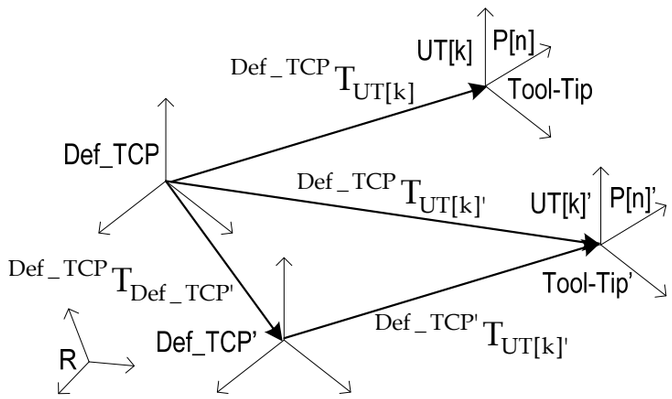


Fig. 2. Shifting a robot point through the offset of frame Def_TCP

TP Instructions	Description
1. Tool_Offset Conditions PR[x], UTOOL[k],	Offset value $^{Def\_TCP}T_{Def\_TCP'}$ is stored in a user-specified position register PR[x].
2. J P[n] 100% Fine Tool_Offset	The "Offset" option in motion instruction shifts the existing robot point $P[n]^{R}_{UT[k]}$ into corresponding point $P[n]'^{R}_{UT[k]'}$.

Table 3. UT[k] offset instruction of FANUC TPP language

2.2 Design of Robot User Frame

In the industrial robot system, the robot programmer is able to establish a robot user frame UF[i](x, y, z) relative to frame R and save it in robot system variable UFRAME[i]. A defined UF[i] can be selected within a robot program or from the robot teach pendant. The robot system uses the current active UF[i] to record robot point $P[n]^{UF[i]}_{UT[k]}$ as shown in Eq. (11) and

cannot move the robot to any robot point $P[m]^{UF[j]}_{UT[k]}$ that is taught with a UF[j] different from UF[i] (i.e. $j \neq i$).

$$P[n]^{UF[i]}_{UT[k]} = {}^{UF[i]}T_{UT[k]}. \tag{11}$$

It is obvious that a robot point $P[n]^{R}_{Def\_TCP}$ in Eq. (1) or Eq. (3) can be taught with different UT [k] and UF[i], thus, represented in different Cartesian coordinates in the robot system as shown in Eq. (12)

$$P[n]^{UF[i]}_{UT[k]} = ({}^{R}T_{UF[i]})^{-1} \times P[n]^{R}_{Def\_TCP} \times {}^{Def\_TCP}T_{UT[k]}. \tag{12}$$

However, the joint representation of a $P[n]^{R}_{Def\_TCP}$ uniquely defines the robot pose.

The robot programmer can directly define a UF[i] with a known robot position measured in frame R. Table 4 shows the UF[i] setup instructions of FANUC TPP language.

FANUC TPP Instructions	Description
1. UFRAME[i]=PR[x]	Assign the value of a robot position register PR[x] to UF[i]
2. UFRAME[i]=LPOS	Assign the current coordinates of frame Def_TCP to UF[i]
3. UFRAME_NUM= i	Set UF[i] to be active in the robot system

Table 4. UF[i] setup instructions of FANUC TPP language

However, to define a UF[i] at a position whose coordinates (x, y, z, w, p, r) in frame R is unknown, the robot programmer needs to follow the UF Setup procedure provided by the robot system and teach four specially defined points $P[n]^{R}_{UT[k]}$ (for n = 1, 2, … 4) where UT[k] represents the tool-tip point of a pointer. In this method as shown in Fig. 3, the location coordinates (x, y, z) of P[4] (i.e. the system-origin point) defines the actual UF[i] origin. The robot system defines the x-, y- and z-axes of frame UF[i] through three mutually perpendicular unit vectors a, b, and c as shown in Eq. (13)

$$\vec{c} = \vec{a} \times \vec{b}, \tag{13}$$

where the coordinates of vectors a and b are determined by the location coordinates (x, y, z) of robot points P[1] (i.e. the positive x-direction point), P[2] (i.e. the positive y-direction point), and P[3] (i.e. the system orient-origin point) in R frame as shown in Fig. 3.

With a taught UF[i], the robot programmer is able to teach a group of robot points relative to it and shift the taught points through its offset value. Fig. 4 shows the method for shifting a taught robot point $P[n]^{UF[i]}_{UT[k]}$ with the offset of UF[i].

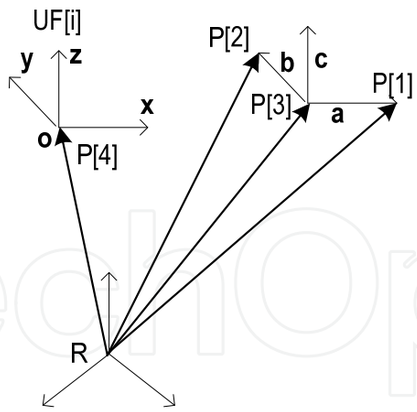


Fig. 3. The four-point method in teaching a UF[i]

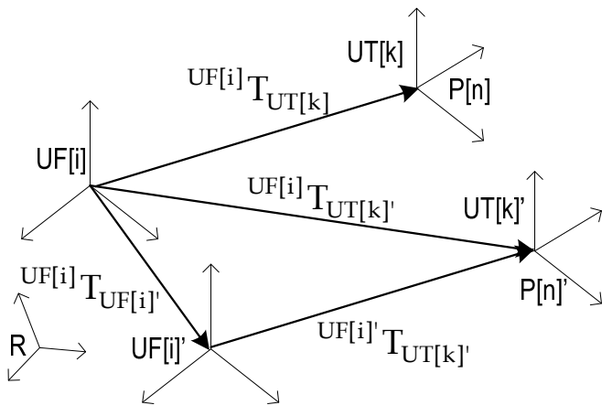


Fig. 4. Shifting a robot point through the offset of UF[i]

Assume that transformation ${}^{UF[i]}T_{UT[k]}$ represents a taught robot point P[n] and remains unchanged when P[n] shifts to P[n]' as shown in Eq. (14)

$$\begin{aligned} {}^{UF[i]}T_{UT[k]} &= {}^{UF[i]'}T_{UT[k]'} \\ \text{or} \quad P[n]_{UT[k]}^{UF[i]} &= P[n]'^{UF[i]'}_{UT[k]'}, \end{aligned} \tag{14}$$

where frame UF[i]' represents the position of frame UF[i] after P[n] becomes P[n]'. Also, assume that transformation ${}^{UF[i]}T_{UF[i]'}$ represents the position change of UF[i]' relative to UF[i], thus, transformation ${}^{UF[i]}T_{UT[k]}$ (or robot point $P[n]_{UT[k]}^{UF[i]}$) can be converted (or shifted) to ${}^{UF[i]}T_{UT[k]'}$ (or $P[n]'^{UF[i]}_{UT[k]'}$) as shown in Eq. (15)

$$\begin{aligned} {}^{UF[i]}T_{UT[k]'} &= {}^{UF[i]}T_{UF[i]'} \times {}^{UF[i]'}T_{UT[k]'} \\ \text{or} \quad P[n]'^{UF[i]}_{UT[k]'} &= {}^{UF[i]}T_{UF[i]'} \times P[n]_{UT[k]}^{UF[i]}. \end{aligned} \tag{15}$$

Usually, the industrial robot system implements Eq. (14) and Eq. (15) as both a system utility function and a program instruction. As a system utility function, offset ${}^{UF[i]}T_{UF[i]'}$ changes the current UF[i] of a taught robot point P[n] into a different UF[i]' without changing its Cartesian coordinates in frame R. As a program instruction, ${}^{UF[i]}T_{UF[i]'}$ shifts a taught robot point $P[n]^{UF[i]}_{UT[k]}$ into the corresponding point $P[n]'^{UF[i]}_{UT[k]'}$ without changing its original UF[i]. Table 5 shows the UF[i] offset instruction of FANUC TPP language for Eq. (15).

FANUC TPP Instructions	Description
3. Offset Conditions PR[x], UFRAME(i),	Offset value ${}^{UF[i]}T_{UF[i]'}$ is stored in a user-specified position register PR[x].
4. J P[n] 100% Fine Offset	The "Offset" option in motion instruction shifts the existing robot point $P[n]^{UF[i]}_{UT[k]}$ into corresponding point $P[n]'^{UF[i]}_{UT[k]'}$.

Table 5. UF[i] offset instruction of FANUC TPP language

A robot point $P[n]^{UF[i]}_{UT[k]}$ can also be shifted by the offset value stored in a robot position register PR[x]. In the industrial robot system, a PR[x] functions to hold the robot position data such as a robot point P[n], the current value of frame Def_TCP (LPOS), or the value of a user-defined robot frame. Different robot languages provide different instructions for manipulating PR[x]. When a PR[x] is taught in a motion instruction, its Cartesian coordinates are defined relative to the current active UT[k] and UF[i] in the robot system. Unlike a taught robot point $P[n]^{UF[i]}_{UT[k]}$ whose UT[k] and UF[i] cannot be changed in a robot program, the UT[k] and UF[i] of a taught PR[x] are always the current active ones in the robot program. This feature allows the robot programmer to use the Cartesian coordinates of a PR[x] as the offset of the current active UF[i] (i.e. ${}^{UF[i]}T_{UF[i]'}$) in the robot program for shifting the robot points as discussed above.

3. Creating Robot Points through Robot Vision System

Within the robot workspace the position of an object frame Obj[n] can be measured relative to a robot UF[i] through sensing systems such as a machine vision system. Methods for integrating vision systems into industrial robot systems have been developed for many years (Connolly, 2008; Nguyen, 2000). The utilized technology includes image processing, system calibration, and reference frame transformations (Golnabi & Asadpour, 2007; Motta et al., 2001). To use the vision measurement in the robot system, the robot programmer must establish a vision frame Vis[i](x, y, z) in the vision system and a robot $UF[i]_{cal}$(x, y, z) in the robot system, and make the two frames exactly coincident. Under this condition, a vision measurement represents a robot point as shown in Eq. (16)

$$ {}^{Vis[i]}T_{Obj[n]} = {}^{UF[i]cal}T_{Obj[n]} = P[n]^{UF[i]cal}_{UT[k]} . \quad (16) $$

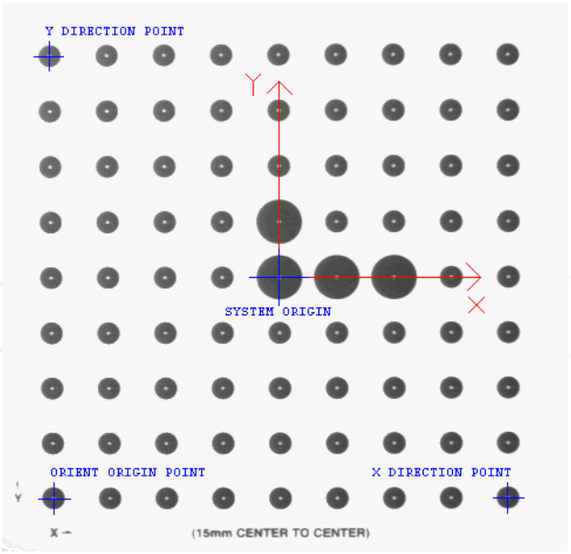
3.1 Vision System Setup

A two-dimensional (2D) robot vision system is able to use the 2D view image taken from a single camera to identify a user-specified object and measure its position coordinates (x, y, roll) for the robot system. The process of vision camera calibration establishes the vision frame Vis[i] (x, y) and the position value (x, y) of a pixel in frame Vis[i]. The robot programmer starts the vision calibration by adjusting both the position and focus of the camera for a completely view of a special grid sheet as shown in Figure 5a**.** The final camera position for the grid view is the "camera-calibration position" $P[n]_{cal}$. During the vision calibration, the vision software uses the images of the large circles to define the x- and y-axes of frame Vis[i] and the small circles to define the pixel value. The process also establishes the camera view plane that is parallel to the grid sheet as shown in Figure 5b. The functions of a geometric locator provided by the vision system allow the robot programmer to define the user-specified searching window, object pattern, and reference frame Obj of the object pattern. After the vision calibration, the vision system is able to identify an object that matches the trained object pattern appeared on the camera view picture and measure position coordinates (x, y, roll) of the object at position Obj[n] as transformation ${}^{Vis[i]}T_{Obj[n]}$.

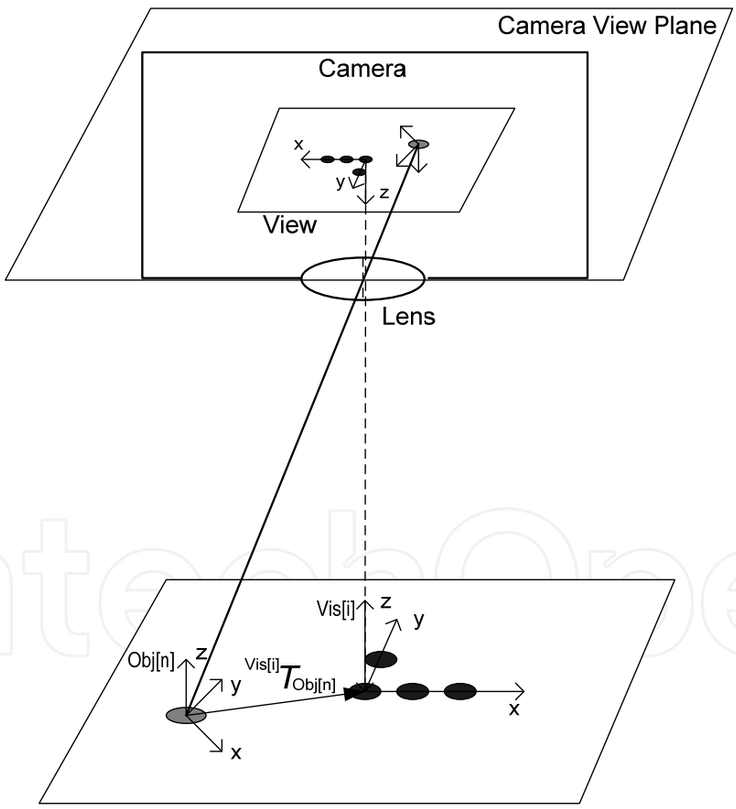
3.2 Integration of Vision "Eye" and Robot "Hand"

To establish a robot user frame $UF[i]_{cal}$ and make it coincident with frame Vis[i], the robot programmer must follow the robot UF Setup procedure and teach four points from the same grid sheet this is at the same position in the vision calibration. The four points are the system origin point, the X and Y direction points, and the orient origin point of the grid sheet as shown in Fig. 5a.

In a "fixed-camera" vision application, the camera must be mounted at the camera-calibration position $P[n]_{cal}$ that is fixed with respect to the robot R frame. Because frame Vis[i] is coincident with frame $UF[i]_{cal}$ when the camera is at $P[n]_{cal}$, the vision measurement ${}^{Vis}T_{Obj[n]}$=(x, y, roll) to a vision-identified object at position Obj[n] actually represents the same coordinates of the object in $UF[i]_{cal}$ as shown in Eq. (16). With additional values of z, pitch, and yaw that can be either specified by the robot programmer or measured by a laser sensor in a 3D vision system, ${}^{Vis[i]}T_{Obj[n]}$ can be used as a robot point $P[n]^{UF[i]_{Cal}}_{UT[k]}$ in the robot program. However, after reaching to vision-defined point $P[n]^{UF[i]_{Cal}}_{UT[k]}$,the robot system cannot perform the robot motions with the robot points that are taught via the same vision-identified object located at a different position Obj[m] (i.e. $m \neq n$).



(a) Camera calibration grid sheet



(b) Vision measurement

Fig. 5. Vision system setup

To reuse all pre-taught robot points in the robot program for the vision-identified object at a different position, the robot programmer must set up the vision system so that it can

determine the position offset of frame $UF[i]_{cal}$ (i.e. ${}^{UF[i]cal}T_{UF[i]'cal}$) with two vision measurements ${}^{Vis}T_{Obj[n]}$ and ${}^{Vis}T_{Obj[m]}$ as shown in Fig. 6 and Eq. (17)

$$ {}^{UF[i]cal}T_{UF[i]'cal} = {}^{Vis[i]}T_{Obj[m]} \times ({}^{Vis[i]'}T_{Obj[n]})^{-1}, $$

and (17)

$$ {}^{UF[i]cal}T_{Obj[n]} = {}^{Vis[i]}T_{Obj[n]} = {}^{Vis[i]'}T_{OBJ[m]} = {}^{UF[i]'cal}T_{Obj[m]}, $$

where frames Vis[i]' and UF[i]'$_{cal}$ represent the positions of frames Vis[i] and $UF[i]_{cal}$ after object position Obj[n] changes to Obj[m]. Usually, the vision system obtains ${}^{Vis[i]}T_{Obj[n]}$ during the vision setup and acquires ${}^{Vis[i]}T_{Obj[m]}$ when the camera takes the actual view picture for the object.

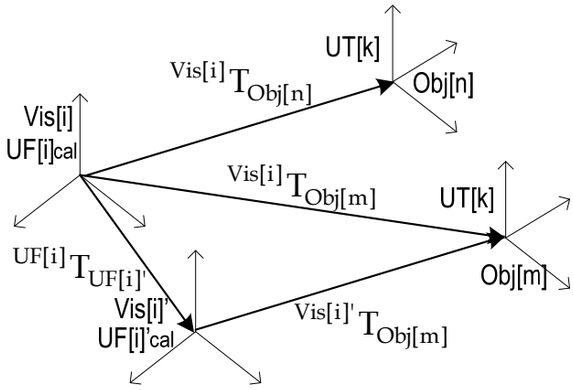


Fig. 6. Determining the offset of frame $UF[i]_{cal}$ through two vision measurements

In a "mobile-camera" vision application, the camera can be attached to the robot's wrist faceplate and moved by the robot on the camera view plane. In this case, frames $UF[i]_{cal}$ and Vis[i] are not coincident each other when camera view position $P[m]_{vie}$ is not at $P[n]_{cal}$. Thus, vision measurement ${}^{Vis[i]}T_{Obj[m]}$ obtained at $P[m]_{vie}$ cannot be used for determining ${}^{UF[i]cal}T_{UF[i]'cal}$ in Eq. (17) directly. However, it is noticed that frame Vis[i] is fixed in frame Def_TCP and its position coordinates can be determined in Eq. (18) as shown in Fig. 7

$$ {}^{Def\_TCP}T_{Vis[i]} = ({}^{R}T_{Def\_TCP})^{-1} \times {}^{R}T_{UF[i]_{cal}}, \tag{18} $$

where transformations ${}^{R}T_{UF[i]cal}$ and ${}^{R}T_{Def\_TCP}$ are uploaded from the robot system when the robot-mounted camera is at $P[n]_{cal}$ during the vision setup. With vision-determined ${}^{Def\_TCP}T_{Vis[i]}$, vision measurement ${}^{Vis[i]}T_{Obj[m]}$ can be transformed into ${}^{UF[i]}T_{Obj[m]}$ for the robot system in Eq. (19) if frame Def_TCP is used as frame $UF[i]_{cal}$ (i.e. $UF[i]_{cal}$ = Def_TCP) in the robot program as shown in Fig. 7.

$$ {}^{UF[i]cal}T_{Obj[m]} = {}^{Def\_TCP}T_{Obj[m]} = {}^{Def\_TCP}T_{Vis[i]} \times {}^{Vis[i]}T_{Obj[m]}. \tag{19} $$

By substituting Eq. (19) into Eq. (17), frame offset ${}^{UF[i]cal}T_{UF[i]'cal}$ can be determined in Eq. (20)

$$ {}^{UF[i]cal}T_{UF[i]'cal} = {}^{Def\_TCP}T_{Vis[i]} \times {}^{Vis[i]}T_{Obj[m]} \times ({}^{Vis[i]}T_{Obj[n]})^{-1} $$

and (20)

$$ {}^{UF[i]cal}T_{Obj[n]} = {}^{Vis[i]}T_{Obj[n]} = {}^{Vis[i]'}T_{Obj[m]} = {}^{UF[i]'cal}T_{Obj[m]'} $$

where frames Vis[i]′ and UF[i]′$_{cal}$ represent positions of frames Vis[i] and UF[i]$_{cal}$ after object position Obj[n] changes to Obj[m].

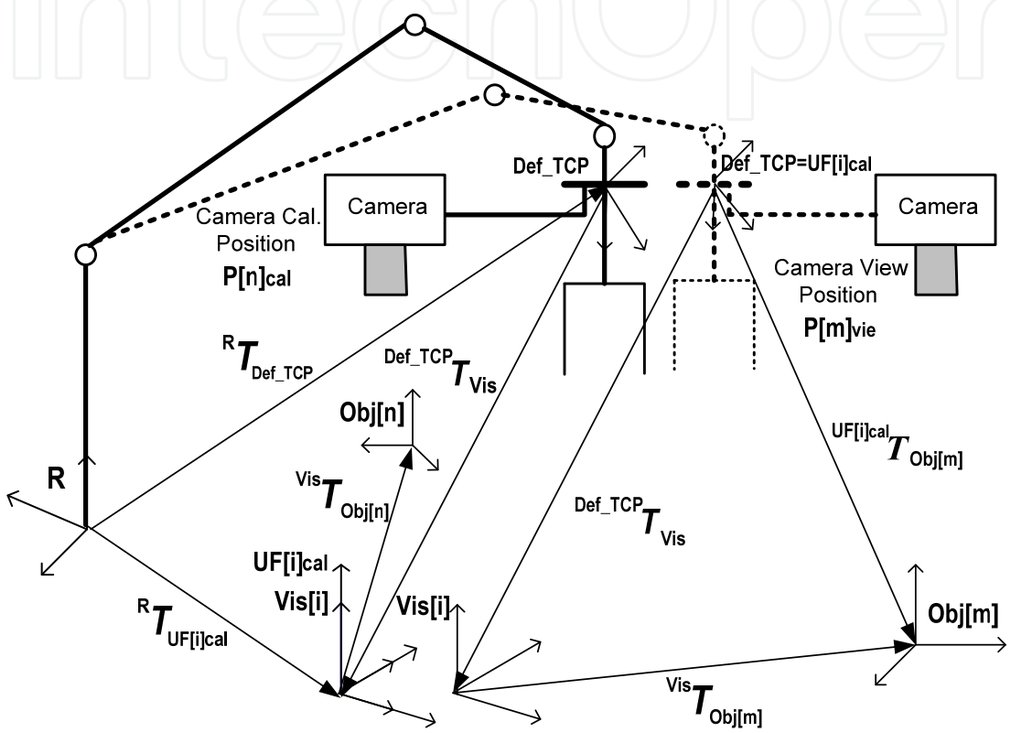


Fig. 7. Frame transformations in mobile-camera application

With vision-determined ${}^{UF[i]cal}T_{UF[i]'cal}$ in Eq. (17) (for fixed-camera) or Eq. (20) (for mobilecamera), the robot programmer is able to apply Eq. (15) for shifting all pre-taught robot points $P[n]^{UF[i]cal}_{UT[k]}$ into $P[n]'^{UF[i]cal}_{UT[k]}$ for the vision-identified object at position Obj[m] as shown in Eq. (21)

$$ P[n]'^{UF[i]cal}_{UT[k]} = {}^{UF[i]cal}T_{UF[i]'cal} \times P[n]^{UF[i]'cal}_{UT[k]} $$

and (21)

$$ P[n]'^{UF[i]'cal}_{UT[k]} = P[n]^{UF[i]cal}_{UT[k]} . $$

Table 6 shows the FANUC TP program used in a fixed-camera FANUC vision application. The program calculates "vision offset" ${}^{UF[i]cal}T_{UF[i]'cal}$ in Eq. (17), sends it to user-specified robot position register PR[x], and transforms robot point $P[n]^{UF[i]cal}_{UT[k]}$ in Eq. (21).

FANUC TP Program	Description
1: R[1] = 0;	Clear robot register R[1] which is used as the indicator for vision "Snap & Find" operation.
2: VisLOC Snap & Find ('2d Single', 2);	Acquire ${}^{Vis}T_{OBJ[m]}$ from snapshot view picture '2d single', find vision-measured offset ${}^{UF[i]cal}T_{UF[i]'cal}$, and send it to robot position register PR[1].
3: WAIT R[1] <> 0;	Wait until the VisLOC vision system sets R[1] to '1' for a successful vision "Snap & Find" operation.
4: IF R[1] <> 1, JMP LBL[99]	Jump out of the program if the vision system cannot set R[1] as '1'.
5: OFFSET CONDITION PR[1], UFRAME[i]$_{cal}$;	Apply ${}^{UF[i]cal}T_{UF[i]'cal}$ as Offset Condition.
6: J P[n] 50% FINE OFFSET;	Transforms robot point $P[n]^{UF[i]cal}_{UT[k]}$ by ${}^{UF[i]cal}T_{UF[i]'cal}$.

Table 6. FANUC TP program used in a fixed-camera FANUC vision application

4. Creating Robot Points through Robot Simulation System

With the today's robot simulation technology a robot programmer may also utilize the robot simulation software to program the motions and actions of a real robot offline in a virtual and interactive 3D design environment. Among many robot simulation software packages, the DELMIA Interactive Graphics Robot Instruction Program (IGRIP) provides the robot programmers with the most comprehensive and generic simulation functions, industrial robot models, CAD data translators, and robot program translators (Cheng, 2003; Connolly, 2006) .

In IGRIP, a simulation design starts with building the 3D device models (or Device) based on the geometry, joints, kinematics of the corresponding real devices such as a robot and its peripheral equipment. The base frame B[i](x, y, z) of a retrieved Device defines its position in the simulation workcell (or Workcell). With all required Devices in the Workcell, the robot programmer is able to create virtual robot points called tag points and program the desired motions and actions of the robot Device and end-effector Device in robot simulation language. Executing the Device simulation programs allows the robot programmer to verify the performance of the robot Device in the Workcell. After the tag points are adjusted relative to the position of the corresponding robot in the real robot workcell through conducting the simulation calibration, the simulation robot program can be downloaded to the real robot controller for execution. Comparing to the conventional online robot programming, the true robot offline programming provides several advantages in terms of the improved robot workcell performance and reduced robot downtime.

4.1 Creation of Virtual Robot Points

A tag point Tag[n] is created as a Cartesian frame and attached to the base frame B[i] of a user-selected Device in the Workcell. Mathematically, the Tag[n] position is measured in frame B[i] as frame transformation ${}^{B[i]}T_{Tag[n]}$ and can be manipulated through functions of Selection, Translation, Rotation, and Snap. During robot simulation, the motion instruction in the robot simulation program is able to move frame Def_TCP (or UT[k]) of the robot Device to coincide a Tag[n] only if it is within the robot's workspace and not a robot's singularity. The procedures for creating and manipulating tag points in IGRIP are:

Step 1. Create a tag path and attach it to frame B[i] of a selected Device.

Step 2. Create tag points Tag[n] (for n = 1, 2, … m) one at a time in the created path.

Step 3. Manipulate a Tag[n] in the Workcell. Besides manipulation functions of selection, translation, and/or rotation, the "snap" function allows the programmer to place a Tag[n] to the vertex, edge, frame, curve, and surface of any Device in the Workcell. Constraints and options can also be set up for a specific snap function. For example, if the "center" option is chosen, a Tag[n] will be snapped on the "center" of the geometric entities such as line, edge, polygon, etc. If a Tag[n] is required to snap on "surface," the parameter "approach axis" must be set up to determine which axis of Tag[n] will be aligned with the surface normal vector.

4.2 Accuracy Enhancement of Virtual Robot Points

It is obvious that inevitable differences exist between the real robot wokcell and the simulated robot Workcell because of the manufacturing tolerance and dimension variation of the corresponding components. Therefore, it is not feasible to directly download tag point Tag[n] to the actual robot controller for execution. Instead, the robot programmer must apply the simulation calibration functions to adjust the tag points with respect to a number of robot points uploaded from the real robot workcell. The two commonly used calibration methods are calibrating frame UT[k] of a robot Device and calibrating frame B[i] of a Device that attaches Tag[n]. The underlying principles of these methods are the same with the design of robot UT and UF frames as introduced in section 2.1 and 2.2. For example, assume that the UT[k]' of the robot end-effector Device is not exactly the same with the UT[k] of the actual robot end-effector prior to UT[k] calibration. To determine and use the actual UT[k] in the simulation Workcell, the programmer needs to teach three non-collinear robot points through UT Frame Setup procedure in the real robot system and upload them into the simulation Workcell so that the simulation system is able to calculate the origin of UT[k] with the "three-point" method as described in Eq. (5) and Eq. (6) in section 2.1. With the calibrated UT[k] and the assumption that the robot Device is exactly the same as the real robot, the UT[k] position relative to the R frame (${}^{R}T_{UT[k]}$) of a robot Device in the simulation Workcell is exactly the same as the corresponding one in the real robot workcell. Also, prior to frame B[i] calibration, the Tag[n] position relative to frame R of a robot Device (${}^{R}T_{Tag[n]}$) may not be the same as the corresponding one in the real robot workcell. In this case, the Device that attaches Tag[n] serves as a "fixture" Device. Thus, the programmer may define a robot UF[i] frame by teaching (or create) three or six robot points (or tag points) on the features of the real "fixture" device (or "fixture" Device) in the real workcell (or the simulation Workcell). Coinciding the created UF tag points in the simulation Workcell with

the corresponding uploaded real robot points results in calibrating the position of frame B[i] of the "fixture" Device and the Tag[n] attached to it.

5. Transferring Robot Points to Identical Robots

In industrial robot applications, there are often the cases in which the robot programmer must be able to quickly and reliably change the existing robot points in the robot program so that they can be accurate to the slight changes of components in the existing or identical robot workcell. Different methods have been developed for measuring the dimensional difference of the similar components in the robot workcell and using it to convert the robot points in the existing robot programs. For example, as introduced in section 2.1 and 2.2, the robot programmer can measure the positional variations of two similar tool-tip points and workpieces in the real robot workcell through the offsets of UT[k] and UF[i], and compensate the pre-taught robot points with either the robot system utility function or the robot program instruction. However, if the dimensional difference exists between two identical robots, an external calibration system must be used for identifying the robots' difference so that the taught robot points P[n] for one robot system can be transferred to the identical one. The process is called the robot calibration, which consists of four steps (Cheng, 2007; Motta et al, 2001). The first step is to teach specially defined robot points P[n]. The second step is to "physically" measure the taught P[n] with an appropriate external measurement device such as laser interferometry, stereo vision, or mechanical "string pull" devices, etc. The third step is to calculate the relevant actual parameters of the robot frames through a specific mathematical solution.

The Dynalog DynaCal system is a complete robot calibration system that is able to identify the parameters of robot joint frames, UT[k], and UF[i] in two "identical" robot workcells, and compensate the existing robot points so that they can be download to the identical robot system for execution. Among its hardware components, the DynaCal measurement device defines its own measurement frame through a precise base adaptor mounted at an alignment point. It uses a high resolution, low inertia optical encoder to constantly measure the extension of the cable that is connected to the tool-tip point of the robot's end-effector through a DynaCal TCP adaptor, and sends the encoder measurements to the Window-based DynaCal software for the identification of the robot parameters.

Prior to the robot calibration, the robot programmer needs to conduct the calibration experiment in which a developed robot calibration program moves the robot Def_TCP frame to a set of taught robot calibration points. Depending on the required accuracy, at least 30 calibration points are required. It is also important to select robot calibration points that are able to move each robot joint as much as possible in order to "excite" its calibration parameters. The dimensional difference of the robot joint parameters is then determined through a specific mathematical solution such as the standard non-linear least squares optimization. Theoretically, the existing robot kinematics model can be modified with the identified robot parameters. However, due to the difficulties in directly modifying the kinematic parameters of an actual robot controller, the external calibration system compensates the corresponding joint values of all robot points in the existing robot program by solving the robot's inverse kinematics equations with the identified robot joint parameters.

In DynaCal UT[k] calibration, the programmer needs to specify at least three non-collinear measurement points on the robot end-effector and input their locations relative to the desired tool-tip point in the DynaCal system during the DynaCal robot calibration. However, when only the UT[k] origin needs to be calibrated, one measurement point on the end-effector suffices and choosing the measurement point at the desired tool-tip point further simplifies the process because its location relative to the desired tool-tip point is then simply zero. In DynaCal UF[i] calibration, the programmer needs to mount the DynaCal measurement device at three (or four) non-collinear alignment points on a fixture during the DynaCal robot calibration. The position of each alignment point relative to the robot R frame is measured through the DynaCal cable and the TCP adaptor at the calibrated UT[k]. The DynaCal software uses the measurements to determine the transformation between the $UF[i]_{Fix}$ on the fixture and the robot R frame, denoted as ${}^{R}T_{UF[i]Fix}$. With the identified values of frames UT[k] and $UF[i]_{Fix}$ in the original robot workcell and the values of UT[k]' and $UF[i]'_{Fix}$ in the "identical" robot workcell, offsets UF and UT can be determined and the robot points P[n] used in the original robot cell can be converted into the corresponding ones for the "identical" robot cell with the methods as introduced in sections 2.1 and 2.2.

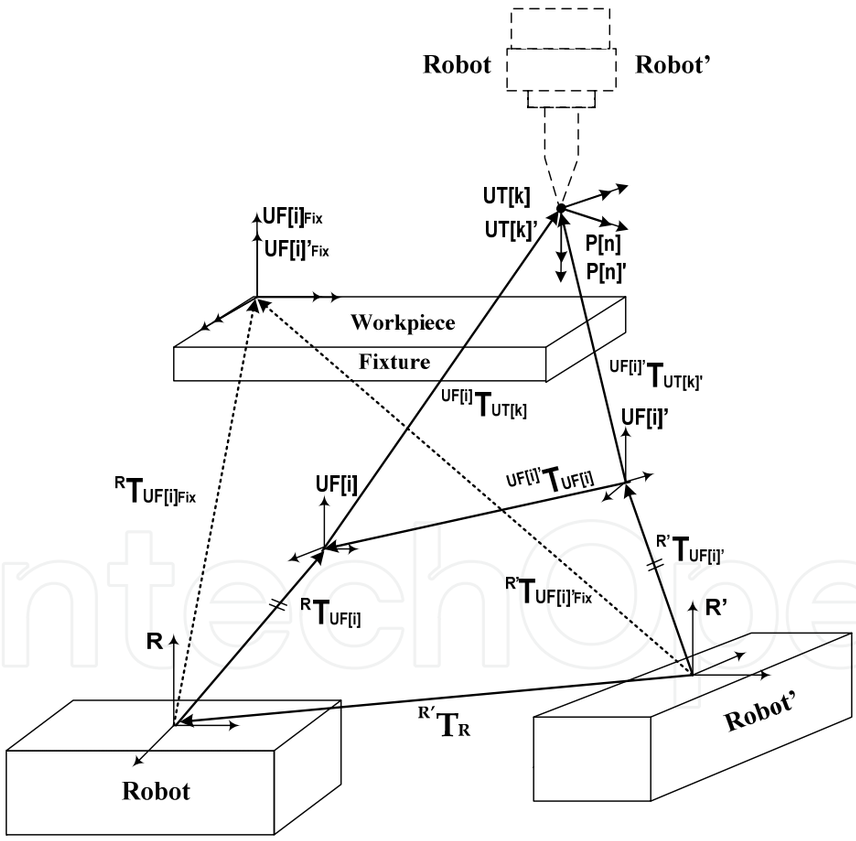


Fig. 8. Determining the offset of UF[i] in two identical robot workcells through robot calibration system

The following frame transformation equations show the method for determining the robot offset ${}^{UF[i]'}T_{UF[i]}$ in two identical robot workcells through calibrated values of $UF[i]_{Fix}$ and

$UF[i]'_{Fix}$ as shown in Fig. 8. Given that the coincidence of $UF[i]_{Fix}$ and $UF[i]'_{Fix}$ represents a commonly used calibration fixture in two "identical" robot workcells, the transformation between two robot base frames R' and R can be calculated in Eq. (22)

$$^{R'}T_{R} = {}^{R'}T_{UF[i]'_{Fix}} \times ({}^{R}T_{UF[i]_{Fix}})^{-1}. \tag{22}$$

It is also possible to make transformation $^{R'}T_{UF[i]'}$ equal to transformation $^{R}T_{UF[i]}$ as shown in Eq. (23)

$$^{R'}T_{UF[i]'} = {}^{R}T_{UF[i]}, \tag{23}$$

where frames UF[i] and UF[i]' are used for recording robot points P[n] and P[n]' in the two "identical" robot workcells, respectively. With Eq. (22) and Eq. (23), robot offset $^{UF[i]'}T_{UF[i]}$ can be calculated in Eq. (24)

$$^{UF[i]'}T_{UF[i]} = ({}^{R'}T_{UF[i]'})^{-1} \times {}^{R'}T_{R} \times {}^{R}T_{UF[i]}. \tag{24}$$

6. Conclusion

Creating accurate robot points is an important task in robot programming. This chapter discussed the advanced techniques used in creating robot points for improving robot operation flexibility and reducing robot production downtime. The theory of robotics shows that an industrial robot system represents a robot point in both Cartesian coordinates and proper joint values. The concepts and procedures of designing accurate robot user tool frame UT[k] and robot user frame UF[i] are essential in teaching robot points. Depending on the selected UT[k] and UF[i], the Cartesian coordinates of a robot point may be different, but the joint values of a robot point always uniquely define the robot pose. Through teaching robot frames UT[k] and UF[i] and measuring their offsets, the robot programmer is able to shift the originally taught robot points for dealing with the position variations of the robot's end-effector and the workpiece. The similar method has also been successfully applied in the robot vision system, the robot simulation, and the robot calibration system. In an integrated robot vision system, the vision frame Vis[i] serves the role of frame UF[i]. The vision measurements to the vision-identified object obtained in either fixed-camera or mobile-camera applications are used for determining the offset of UF[i] for the robot system. In robot simulation, the virtual robot points created in the simulation robot workcell must be adjusted relative to the position of the robot in the real robot workcell. This task can be done by attaching the created virtual robot points to the base frame B[i] of the simulation device that serves the same role of UF[i]. With the uploaded real robot points, the virtual robot points can be adjusted with respect to the determined true frame B[i]. In a robot calibration system, the measuring device establishes frame UF[i] on a common fixture for the workpiece, and the measurement of UF[i] in the identical robot workcell are used to determine the offset of UF[i].

7. References

Cheng, F. S. (2009). Programming Vision-Guided Industrial Robot Operations, *Journal of Engineering Technology*, Vol. 26, No. 1, Spring 2009, pp. 10-15.

Cheng, F. S. (2007). The Method of Recovering TCP Positions in Industrial Robot Production Programs, *Proceedings of 2007 IEEE International Conference on Mechatronics and Automation*, August 2007, pp. 805-810.

Cheng, S. F. (2003). The Simulation Approach for Designing Robotic Workcells, *Journal of Engineering Technology*, Vol. 20, No. 2, Fall 2003, pp. 42-48.

Connolly, C. (2008). Artificial Intelligence and Robotic Hand-Eye Coordination, *Industrial Robot*:*An International Journal,* Vol. 35, No. 6, 2008, pp. 496-503.

Connolly, C. (2007). A New Integrated Robot Vision System from FANUC Robotics, *Industrial Robot: An International Journal,* Vol. 34, No. 2, 2007, pp. 103-106.

Connolly, C. (2006). Delmia Robot Modeling Software Aids Nuclear and Other Industries, *Industrial Robot*:*An International Journal,* Vol. 33, No. 4, 2008, pp. 259-264.

Fanuc Robotics (2007). Teaching Pendant Programming, *R-30iA Mate LR HandlingTool Software Documentation*, Fanuc Robotics America, Inc.

Golnabi, H. & Asadpour, A. (2007). Design and application of industrial machine vision systems, *Robotics and Computer-Integrated Manufacturing*, 23, pp. 630–637.

Motta, J.T.; de Carvalhob, G. C. & McMaster, R.S. (2001). Robot calibration using a 3D vision-based measurement system with a single camera, *Robotics and Computer Integrated Manufacturing*, 17, 2001, pp. 487–497

Nguyen, M. C. (2000), Vision-Based Intelligent Robots, In SPIE: Input/Output and Imaging Technologies II, Vol. 4080, 2000, pp. 41-47.

Niku, S. B. (2001). Robot Kinematics, *Introduction to Robotics: Analysis, Systems, Applications*, pp. 29-67, Prentice Hall. ISBN 0130613096, New Jersey, USA.

Pulkkinen1, T.; Heikkilä1, T.; Sallinen1, M.; Kivikunnas1, S. & Salmi, T. (2008). 2D CAD based robot programming for processing metal profiles in short series, *Proceedings of Manufacturing, International Conference on Control, Automation and Systems 2008*, Oct. 14-17, 2008 in COEX, Seoul, Korea, pp. 157-160.

Rehg, J. A. (2003). Path Control, *Introduction to Robotics in CIM Systems*, 5th Ed. pp. 102-108Prentice Hall, ISBN 0130602434, New Jersey, USA.

Zhang, H.; Chen, H. & Xi, N. (2006). Automated robot programming based on sensor fusion, *Industrial Robot: An International Journal,* Vol. 33, No. 6, 2006, pp. 451-459.

Advances in Robot Manipulators

Edited by Ernest Hall

ISBN 978-953-307-070-4
Hard cover, 678 pages
Publisher InTech
Published online 01, April, 2010
Published in print edition April, 2010

The purpose of this volume is to encourage and inspire the continual invention of robot manipulators for science and the good of humanity. The concepts of artificial intelligence combined with the engineering and technology of feedback control, have great potential for new, useful and exciting machines. The concept of eclecticism for the design, development, simulation and implementation of a real time controller for an intelligent, vision guided robots is now being explored. The dream of an eclectic perceptual, creative controller that can select its own tasks and perform autonomous operations with reliability and dependability is starting to evolve. We have not yet reached this stage but a careful study of the contents will start one on the exciting journey that could lead to many inventions and successful solutions.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen