


PROGRAMMING AND TRAJECTORY PLANNING OF ROBOTS IN INDUSTRIAL ENVIRONMENT



MASTER'S DEGREE THESIS

RELATORE:

Massimo VIOLANTE

CANDIDATO:

Valentina TURCO

SUMMARY

ABSTRACT	7
1. INTRODUCTION	8
1.1. MOTION PLANNING	8
1.1.1. DIRECT POSITION KINEMATIC FUNCTIONS.....	8
1.1.2. INVERSE POSITION KINEMATIC FUNCTIONS.....	10
1.1.3. DIRECT VELOCITY KINEMATIC FUNCTIONS.....	11
1.1.4. INVERSE VELOCITY KINEMATIC FUNCTION.....	14
1.1.5. JOINT SPACE TRAJECTORY PLANNING	15
1.1.6. TASK SPACE TRAJECTORY PLANNING	18
1.2. ANALYSIS OF THE STATE OF ART OF PROGRAMMING LANGUAGES FOR COMAU, FANUC AND ABB	
20	
1.2.1. COMAU TRAJECTORY PLANNING.....	22
1.2.2. FANUC PROGRAMMING LANGUAGE.....	22
2.1.1. ABB PROGRAMMING LANGUAGE.....	23
2. DEFINITION OF THE SYSTEM.....	24
2.1 ROBOT R1	26
2.1.1. I/O CONFIGURATION	28
2.2 ROBOT R2	30
2.2.1. I/O CONFIGURATION	32
2.3 ROBOT R3	34
2.3.1. I/O CONFIGURATION	36
3. PICK AND PLACE TASK.....	39
3.1. ROBOT R1	40
3.1.1. PICK_GRIPPER1_ENTRY_LINE	40
3.1.2. PICK_GRIPPER2_ST5	41
3.1.3. PLACE_GRIPPER1_ST5	42
3.1.4. PLACE_GRIPPER1_NOK1_A_CONVEYOR	42
3.1.5. PLACE_GRIPPER2_ST6	43
3.1.6. PLACE_GRIPPER2_ST7	43
3.1.7. PLACE_GRIPPER2_NOK1_B_CONVEYOR	44
3.2. ROBOT R2	45
3.2.1. PICK1.....	45
3.2.2. PICK2.....	46
3.2.3. PLACE1.....	47
3.2.4. PLACE2.....	48
3.3. ROBOT R3.....	49
3.3.1. PICK_GRIPPER2_ST6	49
3.3.2. PICK_GRIPPER2_ST7	50
3.3.3. PICK_GRIPPER2_NOK_ST9.....	51
3.3.5. PLACE_GRIPPER2_NOK_FC_CONVEYOR.....	52
3.3.6. PLACE_GRIPPER2_NOK_ML_CONVEYOR.....	53
3.3.7. PLACE_GRIPPER1_ST9	53
3.3.8. PLACE_GRIPPER1_NOK_C_CONVEYOR.....	54

4. MOVEMENTS CONFIGURATION..... 55

4.1. ROBOT R1..... 57

4.1.1. SAFEMOVE 57

4.1.2. WZONES AND LIMITS..... 57

4.2. ROBOT R2..... 60

4.2.1. WZONES AND LIMITS..... 60

4.3. ROBOT R3..... 62

4.3.1. SAFEMOVE 62

4.3.2. WZONES AND LIMITS..... 62

5. CONCLUSIONS..... 66

FIGURE SUMMARY

Figure 1: joints and arms definition.....	9
Figure 2: example of Denavit-Hartenberg convention.....	10
Figure 3: example of a spherical and a non spherical wrist	10
Figure 4: computation of the geometric Jacobian	12
Figure 5: trajectory planner.....	15
Figure 6: position, velocity and acceleration profiles.....	16
Figure 7: bang-bang trajectory profiles.....	17
Figure 8: axis-angle interpolation example	19
Figure 9: planar sliding interpolation	19
Figure 10: Euler angle interpolation.....	20
Figure 11: line layout.....	25
Figure 12: R1 task description	26
Figure 13: TCP coordinate system with respect to the base one.....	27
Figure 14: gripper 1 and 2 reference frames with respect to the TCP	27
Figure 15: Inputs configuration	28
Figure 16: Outputs configuration	29
Figure 17: R2 task description	30
Figure 18: TCP reference frame with respect to the base one	31
Figure 19: gripper 1 and 2 reference frame with respect to the TCP.....	31
Figure 20: Inputs configuration	32
Figure 21: Outputs configuration	33
Figure 22: R3 task description	34
Figure 23: TCP reference frame with respect to the base one	35
Figure 24: gripper 1 and 2 reference frames with respect to the TCP	35
Figure 25: input configuration	37
Figure 26: output configuration	38
Figure 27: pick task from entry line performed by gripper 1	41
Figure 28: pick task from ST5 performed by gripper 2.....	41
Figure 29: place task to ST5 performed by gripper 1	42
Figure 30: place task to NOK1_A conveyor performed by gripper 1	42
Figure 31: place task to ST6 performed by gripper 2	43
Figure 32: place task to ST7 performed by gripper 2	43
Figure 33: place task to NOK1_B conveyor performed by gripper 2.....	44
Figure 34: position of rivet 1 computation.....	45
Figure 35: pick1 procedure performed by robot R2.....	46
Figure 36: position of rivet 2 computation.....	46
Figure 37: pick2 procedure performed by robot R2.....	47
Figure 38: place 1 procedure performed by gripper 1	48
Figure 39: place2 procedure performed by robot R2.....	48
Figure 40: pick task from ST6 performed by gripper 2.....	49
Figure 41: pick task from ST7 performed by gripper 2.....	50
Figure 42: pick task from ST8 performed by gripper 1.....	50
Figure 43: pick task from ST9 performed by gripper 2.....	51
Figure 44: place task to ST8 performed by gripper 2	52
Figure 45: place task to NOK_FC conveyor performed by gripper 2.....	52

Figure 46: place task to NOK_ML conveyor performed by gripper 2.....	53
Figure 47: place task to ST9 performed by gripper 1	53
Figure 48: place task to NOK_C conveyor performed by gripper 1.....	54
Figure 49: SafeMove configuration of robot R1.....	57
Figure 50: signals configuration for robot R1.....	57
Figure 51: home position declaration for robot R1.....	57
Figure 52: home position for robot R1	57
Figure 53: WZones definition for robot R1.....	58
Figure 54: WZones for robot R1	58
Figure 55: limits definition for robot R1	59
Figure 56: limits for robot R1.....	59
Figure 57: signals definition definition for robot R2	60
Figure 58: home position definition for robot R2.....	60
Figure 59: home position for robot R2	60
Figure 60: WZones definition for robot R2.....	60
Figure 61: WZones for robot R2	61
Figure 62: limit definition for robot R2.....	61
Figure 63: limit for robot R2	61
Figure 64: SafeMove configuration for robot R3.....	62
Figure 65: signals configuration for robot R3.....	62
Figure 66: home position definition for robot R3	62
Figure 67: home position for robot R3	62
Figure 68: WZones definition for robot R3.....	63
Figure 69: WZones for robot R3	64
Figure 70: limits definition for robot R3	64
Figure 71: limits for robot R3.....	65
Figure 72: configuration parameters of robtarget	66

ABSTRACT

Starting from a brief review about robotics and motion planning, the aim of my work was to program three ABB robots in a predefined industrial environment to perform pick and place tasks, trying to optimize them in terms of cycle time and of trajectories.

In this specific case, we are discussing about the presence of three anthropomorphic robots, with 6 degrees of freedom able to guarantee the maximum possible dexterity used to pick and place objects of different types and weights. Articulated or anthropomorphic means that the shoulder has three revolute joints, the first one vertical and the other two horizontal and parallel. The structure is very similar to human body, with trunk, arm and forearm with a final wrist so it provides the best dexterity and this is why it is used in most of the industrial applications. Task space is a sort of sphere sector, even if there is no a direct correspondence between joint and Cartesian coordinates, but accuracy is not constant inside the task space.

Particularly two of them are equal models of IRB4600, with a maximum capacity of 60 kg and are used to pick and place clutch's mechanisms of a maximum weight of 8 kg, with a gripper of about 32 kg. The third one, instead, an IRB1200, is used to move rivets of few grams, so it is sufficient with a maximum capacity of 5 kg. They both provide a double gripper to optimize the cycle time in order to be able to match the specification of 15s.

The system scenario is composed of 13 stations, positioned in space in a pre-determined way, through which vehicular clutches are balanced, tested and finally marked.

Since the area in which robots move is not regular and there are many obstacles as machines, laser cabinet and manipulators, it was necessary to declare some additional points to allow the movements. Due to this fact, all of them have been performed as linear through the instruction MoveL so that the minimum trajectory was followed. Moreover, since all the robots are equipped with a double gripper, it was necessary to declare two different tools and perform the movements with respect to one of them.

Moreover, all the robots have been configured to meet safety requirements both in hardware and software through the Safe Move and WZones functionalities. Particularly, the two IRB4600 provided both of them due to the critical position inside the line, so their work area has been delimited by the Safe Move, while their movements around the pick and place areas have been managed by the WZones. The third one instead, since it is smaller and placed in a non critical position, provides only the software functionality through which it has been limited while approaching the rived feeder and the manipulator. This way, thanks to the Safe Move, the robot cannot be moved, even manually, outside the declared area, guaranteeing the integrity of the machines and gates around it. From the software point of view, instead, the encumbrance signals are set by the WZones themselves: in fact, every time the robot comes inside a certain area, the encumbrance signal is set to 1, so the PLC knows that for example the movements of the machines are limited while the robot is in that area. Then, to preserve machines integrity, I have defined also upper limits that guarantee that the robot, even in manual mode, cannot reach that altitude, so the mechanic is preserved.

1. INTRODUCTION

The study of the motion of a robot inside an industrial production system starts from the definition of the environment. For the specific case in which a robot is asked to perform a pick and place task, it is needed to take care of all the constraints related to the space in which the robot has to work, as the presence of humans being or of others robots and machines around it, and also to the physical characteristics of its structure as the dexterity, the maximum values of elongation of its arm, the speed and many others.

Besides all the possible features that could have been taken into consideration in this type of problem, I decided to concentrate my attention particularly on three of them which are the motion planning and the related trajectory optimization and finally the programming stage.

1.1. MOTION PLANNING

Motion planning of a robot can be studied from two completely different points of view which are the joint space and the task space.

For joint space is intended the mathematical structure, so the vector space, whose elements are the joint variables $q_i(t)$. As they describe the motion of each joint, they are angles for revolute joints and distances for prismatic ones. This type of planning is quite difficult to be seen in practice, particularly because the relation between the two spaces is not linear, so to have even simple trajectories, as a straight one, in the joint space it is required something very different. The main characteristic of this type of planning is that it is very easy to find the correspondence between the joint space and the task space through the computation of the direct position and velocity KFs. However, in most of the cases it is better to plan the motion in the task space, which is the space of the tool centre point (TCP), the ideal point of the end effectors that the robot moves through space. This way the trajectory we want to see in space is exactly the same trajectory we need to impose to the robot in order to have the desired motion and this is very useful in all that cases where we need that the end effectors of the robot assumes some specific position or velocity at a certain time due, for example, to the environment characteristic or to the interaction with others robot. Since this information must be given to the joints because they are the only actuators of the whole structure and so only through them it is possible to obtain in practice constraints in motion, it is necessary to be able to describe the relation between the two representations. The transformation from the task space to the joint space is inside the computation of the so called inverse kinematic function, which requires the inverse of the functions related to both position and velocity in order to obtain the exact quantities to be given to the joints to obtain the desired motion. This procedure in general is not very easy, particularly because its computational cost is relevant and often it contains some approximations due to the iterations that must be done in order to obtain the final solution.

1.1.1. DIRECT POSITION KINEMATIC FUNCTIONS

Kinematics functions in general can be related to any point of the robot, but it is convenient if we consider them as position and velocity of the TCP. The first step is to define a reference frame for the base, called R_0 , and one for each arm of the robot. The transformation matrix needed to pass from one RF to the other considers 6 elements, divided in 3 translations of the origin and 3 angles for the rotation. However, to avoid to manage 6 parameters and particularly to be able to find a common way to represent the relative position between two RFs, roboticists introduced a large number of conventions, but the most used is the so called

Denavit-Hartenberg convention. Thanks to it, which is valid both for prismatic and revolute joints, the number of parameters is reduced to 4, two associated with translations, and the others to rotations. Between all of them, three are always constant in time because they depend only on the geometry, while the fourth is the joint variable and changes in time because it depends on the relative motion between two successive links.

To define the number of links and joints of the structure, it is necessary to start from the base of the robot, which is defined as link 0. The first joint we find starting from the base is the joint 1, then we find the link 1 and so on until we reach the TCP. In general, because of the convention, link i is between the joint i and the joint $i+1$.

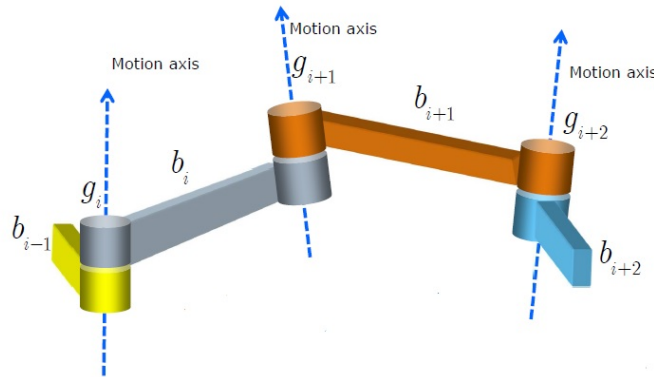


Figure 1: joints and arms definition

In order to place correctly all the RFs, the following rules must be followed:

- The origin of R_i is located on the motion axis g_{i+1} , at the intersection between the common normal to g_i and g_{i+1} . If the two axes intercept, the origin is located at the interception point. If the two axes are parallel, the origin can be located in a point of choice, usually on the arm.
- Unit vector k_i is aligned with the motion axis g_{i+1} , with the positive direction coherent with the positive motion.
- Unit vector i_i is orthogonal to both k_{i-1} and k_i . If they are parallel, i_i belongs to the plane orthogonal to them; the direction chosen by the user. Usually is chosen to lie along the largest symmetry axis of the arm b_i .
- Unit vector j_i completes the right-hand reference frame.

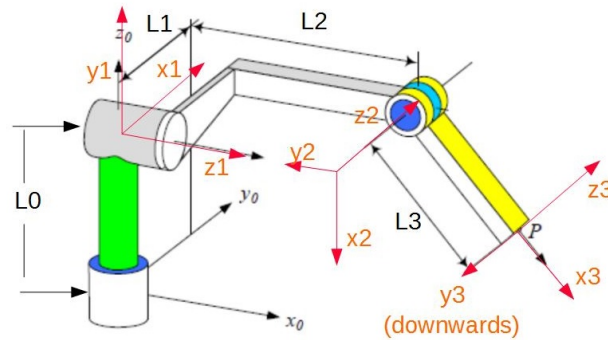
This point two RFs are still undefined, but it is sufficient that:

- For R_0 the unit vector k_0 is aligned with the motion axis g_1 .
- The origin of R_6 is located according to the preferences of the user since no RF will follow, but usually it is on the TCP. The only other condition is that i_6 must be orthogonal to both k_5 and k_6 .

Resuming, the four parameters are:

- d_i : it defines the translation along the motion axis k_{i-1} , between the origin of R_{i-1} and the intersection of the axis defined by k_{i-1} and the axis defined by i_i .
- θ_i : it defines the rotation angle around axis k_{i-1} such that i_{i-1} overlaps i_i . The sign follows the RHR.
- a_i : it defines the minimum signed distance between axis k_{i-1} and k_i along the common normal, measured along i_i .

- α_i : it defines the rotation angle around motion axis i_i such that k_{i-1} overlaps k_i . The sign follow the RHR.



Frame	theta	d	a	alfa
1	$\pi/2$	L_0	0	$\pi/2$
2	$\pi/2$	L_2	0	$-\pi/2$
3	$\pi/2$	L_1	L_3	0

Figure 2: example of Denavit-Hartenberg convention

DH parameters define the transformation from R_{i-1} to R_i , but the joint variable changes depending on the joint. For prismatic joints the parameters θ_i , a_i and α_i are fixed, while $d_i(t)$ is the joint variable. For revolute joints, instead, $\theta_i(t)$ is the joint variable while d_i , a_i and α_i are constant.

Due to the fact that to move from RF to the next it is necessary to perform two translation and two rotations, the total transformation is a roto-translation composed as follow:

$$T = T(R, t) = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix}$$

which, according to the pre-post rules, is composed by:

$$T_i^{i-1} = \text{Trasl}(k, d) \text{Rot}(k, \theta) \text{Trasl}(i, a) \text{Rot}(i, \alpha)$$

In order to obtain the final T_{TCP}^0 all the intermediate transformation matrices must be multiplied together. Finally, it is possible to extract the direct position KF $t_{TCP}^0(q(t))$ and the direct Cartesian orientation KF $R_{TCP}^0(q(t))$, usually expressed using the Euler angles.

1.1.2. INVERSE POSITION KINEMATIC FUNCTIONS

For the computation of the inverse position KF, the problem is much more complex and there is no clear recipe to solve it. A sufficient condition of existence is guarantee in case the wrist of the robot is spherical, which means that the axis of the joints which compose it always meet in a single point, but this doesn't mean that it will be found.

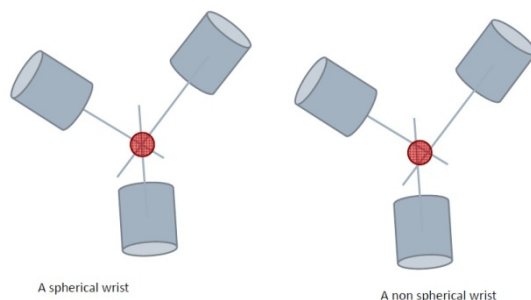


Figure 3: example of a spherical and a non spherical wrist

Due to this fact, several possibilities can be taken into account, as:

- brute force or previous solutions found for similar chains;
- use the inverse velocity KF with a recursive approach which makes this computation easy to be solved;
- use symbolic manipulation programs even if not suggested because very difficult;
- iteratively compute an approximated numerical expression for the non linear equations, as the Newton method:

$$\begin{aligned} p(t) &= f(q(t)) \\ p(t) - f(q(t)) &= 0 \\ \min\{|p(t) - f(q(t))|\} \end{aligned}$$

1.1.3. DIRECT VELOCITY KINEMATIC FUNCTIONS

To compute the direct velocity kinematic function, means to find a relation between the quantities $\dot{q}(t)$ and $\dot{p}(t)$

$$\dot{q}(t) = \begin{pmatrix} \dot{q}_1(t) \\ \dot{q}_2(t) \\ \dot{q}_3(t) \\ \dot{q}_4(t) \\ \dot{q}_5(t) \\ \dot{q}_6(t) \end{pmatrix} \Rightarrow \dot{p}(t) = \begin{pmatrix} \dot{p}_1(t) \\ \dot{p}_2(t) \\ \dot{p}_3(t) \\ \dot{p}_4(t) \\ \dot{p}_5(t) \\ \dot{p}_6(t) \end{pmatrix} = \begin{pmatrix} v(q(t), \dot{q}(t)) \\ \dots \\ \omega(q(t), \dot{q}(t)) \end{pmatrix}$$

Velocity kinematics is characterized by Jacobians:

$$\dot{p}(t) = J(q(t)) * \dot{q}(t)$$

even if it is necessary to make a distinction between the derivative of the angles and the angular velocity, so there are two types of Jacobians. If we consider the angular velocity as a vector, we speak about geometric Jacobian J_g and the relation becomes:

$$v_p(t) = \begin{pmatrix} \dot{x} \\ \omega \end{pmatrix} = J_g \dot{q}$$

In general it is not possible to find a vector $u(t)$ as the integral of $\omega(t)$, the only relation between the two quantities depends both on the vector angle $\alpha(t)$, on its derivative and on the skew-symmetric matrix of $u(t)$ as follows:

$$\omega(t) = \dot{\alpha}(t)u(t) + \sin \alpha(t)\dot{u}(t) + (1 - \cos \alpha(t))S(u(t)) \dot{u}(t)$$

The only case in which the formula can be integrated, but this happens very rarely, is when $\dot{u}(t)=0$, because it means that the rotation axis does not change in time.

The derivatives of the angles, instead, are contained inside the analytical Jacobian J_a , but they cannot be considered as a true vector since the vector addition does not hold to the correct value. This time we have that:

$$\dot{p} = \begin{pmatrix} \dot{x} \\ \dot{\alpha} \end{pmatrix} = J_a * \dot{q}$$

which, because of its definition, it is possible to be integrated as:

$$\int \dot{p}(\tau) d\tau = \int \begin{pmatrix} \dot{x} \\ \dot{\alpha} \end{pmatrix} d\tau = \begin{pmatrix} x(t) \\ \alpha(t) \end{pmatrix}$$

In conclusion, the geometric Jacobian is used every time the physical interpretation of the rotation velocity is needed, while the analytical is adopted when it is necessary to treat differential quantities in the task space.

With this distinction, if two points $q(t_{k+1})$ and $q(t_k)$ are sufficiently near so that it is possible to make an approximation, the direct velocity KF can be used to compute the joint position using a recursive formula:

$$q(t_{k+1}) = q(t_k) + \dot{q}(t_k) \Delta t$$

where the parameter $\dot{q}(t_k)$ can be computed using the geometrical or the analytical Jacobian obtaining:

$$\dot{q}(t_{k+1}) = \dot{q}(t_k) + J_g^{-1}(q(t_k)) v_p(t_k) \Delta t$$

or

$$\dot{q}(t_{k+1}) = \dot{q}(t_k) + J_a^{-1}(q(t_k)) v_p(t_k) \Delta t$$

The geometric Jacobian can be constructed through two steps:

- every link has a reference frame R_i defined according to the DH conventions;
- the position of the origin of R_i is given by:

$$x_i = x_{i-1} + R_{i-1}^0 r_{i-1,i}^{i-1} = x_{i-1} + r_{i-1,i}^0$$

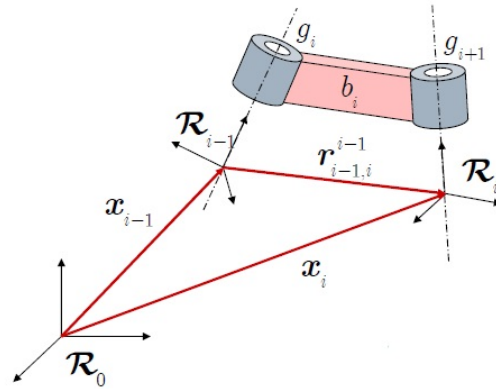


Figure 4: computation of the geometric Jacobian

doing the derivative with respect to time, we obtain:

$$\dot{x}_i = \dot{x}_{i-1} + R_{i-1}^0 \dot{r}_{i-1,i}^{i-1} + \omega_{i-1} \times R_{i-1}^0 r_{i-1,i}^{i-1} = \dot{x}_{i-1} + v_{i-1,i} + \omega_{i-1} \times r_{i-1,i}^0$$

where the second term represents the linear velocity of R_i with respect to R_{i-1} , while the third is the angular velocity of R_{i-1} .

Knowing that:

$$\dot{R} = S(\omega)R = \omega \times R$$

If we derive the composition of two rotations we obtain:

$$\begin{aligned} R_i^0 &= R_{i-1}^0 R_i^{i-1} \\ \dot{R}_i^0 &= \dot{R}_{i-1}^0 R_i^{i-1} + R_{i-1}^0 \dot{R}_i^{i-1} \\ &= S(\omega_{i-1}) R_{i-1}^0 R_i^{i-1} + R_{i-1}^0 S(\omega_{i-1,i}) R_i^{i-1} \\ &= S(\omega_{i-1}) R_{i-1}^0 R_i^{i-1} + S(R_{i-1}^0 \omega_{i-1,i}) R_{i-1}^0 R_i^{i-1} \\ &= [S(\omega_{i-1}) + S(R_{i-1}^0 \omega_{i-1,i})] R_i^0 \equiv S(\omega_i) R_i^0 \end{aligned}$$

Hence the angular velocity of RF_i in RF_0 is equal to the angular velocity of RF_{i-1} in RF_0 plus the angular velocity of RF_i with respect to RF_{i-1} in RF_{i-1} :

$$\omega_i = \omega_{i-1} + R_{i-1}^0 \omega_{i-1,i}$$

The geometric Jacobian can also be decomposed in two parts, one linear containing the contributes to the linear velocity of TCP, the other angular containing instead the contributes to the angular velocity of the TCP:

$$v = \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix} = J_g(q)\dot{q} = \begin{bmatrix} J_{L,1} & J_{L,2} & \dots & J_{L,n} \\ J_{A,1} & J_{A,2} & \dots & J_{A,n} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dots \\ \dot{q}_n \end{bmatrix}$$

More in detail we have that:

$$\dot{x} = \sum_{i=1}^n J_{L,i} \dot{q}_i$$

$$\omega = \sum_{i=1}^n J_{A,i} \dot{q}_i$$

Depending on the type of joint we are considering, it is possible to know in advance what will be the values of the analytical and of the geometrical Jacobian, both represented with respect to R_0 . Moreover we know also that $r_{i-1,TCP}$ is the vector that represents $(x_{TCP} - x_{i-1})$ in R_0 .

	$J_{L,i}$	$J_{A,i}$
Prismatic	k_{i-1}	0
Revolute	$k_{i-1} \times r_{i-1,TCP}$	k_{i-1}

For what concerns the analytical Jacobian, instead, we know that its first three lines (related to the linear representation) are equal to the ones of the geometrical, while the last three (related to the angular representation) are usually different. To pass from the analytical to the geometrical values of these lines, it is necessary to choose the angle representation. In general we have that:

$$\omega = T(\alpha)\dot{\alpha}$$

$$J_g(q) = \begin{bmatrix} I & 0 \\ 0 & T(\alpha) \end{bmatrix} J_a(q)$$

If we consider the Euler angles, we obtain:

$$\alpha = \{\varphi, \theta, \psi\}$$

$$T_E(\alpha) = \begin{bmatrix} 0 & \cos\varphi & \sin\varphi\sin\theta \\ 0 & \sin\varphi & -\cos\varphi\sin\theta \\ 1 & 0 & \cos\theta \end{bmatrix}$$

For the RPY angles, instead, we obtain:

$$\alpha = \{\theta_x, \theta_y, \theta_z\}$$

$$T_{RPY}(\alpha) = \begin{bmatrix} \cos\theta_y\cos\theta_z & -\sin\theta_z & 0 \\ \cos\theta_y\sin\theta_z & \cos\theta_z & 0 \\ -\sin\theta_y & 0 & 1 \end{bmatrix}$$

For both the representations, the values of α that zeros the $T(\alpha)$ determinant correspond to an orientation singularity which means that there are geometric angular velocities that cannot be expressed by joint velocities.

For an anthropomorphic robot three singularity conditions exist:

- completely extended or folded arm: it means that the robot cannot work at the maximum stage, but it works well inside the task space;
- wrist centred on the vertical (which doesn't mean always along the k axis);
- wrist singularity

In case we are using Euler wrists, to compute the singular configuration it is sufficient to start from the symbolic matrix:

$$\begin{pmatrix} C_\varphi C_\psi - S_\varphi C_\theta S_\psi & -C_\varphi S_\psi - S_\varphi C_\theta C_\psi & S_\varphi S_\theta \\ S_\varphi C_\psi + C_\varphi C_\theta S_\psi & -S_\varphi S_\psi + C_\varphi C_\theta C_\psi & -C_\varphi S_\theta \\ S_\psi S_\theta & C_\psi S_\theta & C_\theta \end{pmatrix}$$

it is possible to observe that if $c_\theta = 1$, then $\theta = 0$:

$$\begin{pmatrix} C_\varphi C_\psi - S_\varphi S_\psi & -C_\varphi S_\psi - S_\varphi C_\psi & 0 \\ S_\varphi C_\psi + C_\varphi S_\psi & -S_\varphi S_\psi + C_\varphi C_\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Due to the fact that singularity is only related to velocity and never to position, it means that when joint coordinates approach singularity the joint velocities become very large for small finite Cartesian velocities:

$$\dot{q} = J^{-1}(q)\dot{p} = \frac{1}{\det J}\bar{J}\dot{p} \rightarrow \frac{1}{\varepsilon}\bar{J}\dot{p} \rightarrow \infty$$

Near singularity conditions it is not possible to follow a geometric path and at the same time a given velocity profile. This means that is necessary or to reduce the Cartesian velocity and follow the path, or to follow the velocity profile but with an approximated path because in exact singularity conditions nothing can be done, so it is better to avoid them.

1.1.4. INVERSE VELOCITY KINEMATIC FUNCTION

To compute the inverse velocity kinematic function, if the Jacobian is square, full-rank and non singular, it is sufficient to apply the following formula:

$$\dot{q}(t) = J^{-1}(q(t))\dot{p}(t)$$

When the Jacobian is a rectangular full-rank matrix instead, it means that the robot has a redundant robotic arm, but not singular so there are infinite possible solutions. To compute one of them it is possible to use the pseudo-inverse of the Jacobian, obtaining:

$$\dot{q}(t) = J^+(q(t))\dot{p}(t)$$

where, by definition:

$$J^+ = J^T(JJ^T)^{-1}$$

In case the initial position $q(0)$ is known, it is also possible to compute the inverse velocity KF as an integral in continuous time and as an approximation in discrete time, assuming Δt intervals very small:

$$q(t) = q(0) + \int_0^t \dot{q}(\tau) d\tau$$

$$q(t_{k+1}) \approx q(t_k) + \dot{q}(t_k) \Delta t$$

When the Jacobian, square or rectangular, is not full-rank, it is not possible to invert the matrix using the pseudo-inverse, but it is necessary to use the Singular Decomposition of the Jacobian.

1.1.5. JOINT SPACE TRAJECTORY PLANNING

Before starting speaking about trajectory optimization, it is necessary to point out the distinction between the term path and the term trajectory. The path is the geometrical description of the desired set of points in the task space; this means that the control shall keep the TCP on the desired path. For trajectory, instead, it is intended the path and the time law required to follow it, from the starting to the end point.

Usually in industrial environments for pick and place purposes there are constraints related to time so it is asked to minimize the time during which the item is moved from an initial to a final point. But it could be also the case that there are constraints related to trajectory, so it is better to pass through certain points at a certain speed in order to avoid collisions with others robotic arms and to minimize the time spent moving the item so that at the end the performances and the productivity of the whole system are increased.

The trajectory planner of a robot can be seen as a “software node” that, given a desired path, the desired kinematic constraints (maximum speed) and the robot dynamic constraints (maximum acceleration and torque), is able to determine the joint reference samples q_r for the control block.

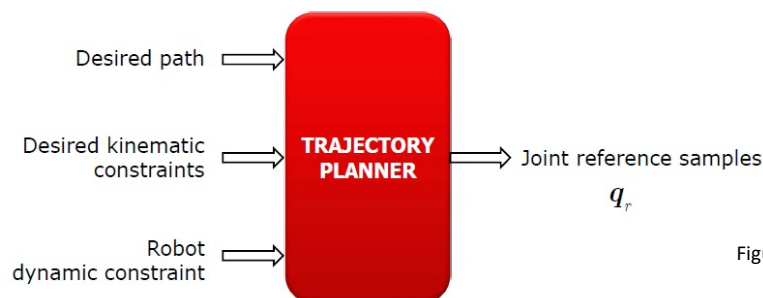


Figure 5: trajectory planner

Having different types of constraint means to have different kinds of information:

- Desired path (task space constraints)
 - a) Initial and final positions
 - b) Initial and final orientations
- Trajectory (time-dependent task space constraints)
 - a) Initial and final velocities
 - b) Initial and final accelerations
 - c) Velocities on a given part of the path (for example constant velocity)
 - d) Accelerations (centrifugal acceleration affecting curvature radius)
 - e) Fly-by points
- Technological constraints (joint space constraints)

- a) Motor maximum velocities
- b) Motor maximum accelerations
- c) Motor temperature

If it is not important to follow a specific path, the trajectory is planned in the joint space implementing a point-to-point (PTP) linear path whose time law directly depends on the maximum velocity and acceleration values of the motors. In this case we obtain a convex combination between the initial and the final values using a unique scalar time-varying quantity called profile abscissa $s(t)$ as follows:

$$\pi'(q(t)) = (1-s(t))q_0 + s(t)q_f = q_0 + s(t)(q_f - q_0) = q_0 + s(t)\Delta q$$

$$0 = s(t_0) \leq s(t) \leq s(t_f) = 1$$

Thanks to this approach we obtain a coordinated motion, which means that the motion of all joints starts and ends at the same time, and we are able to guarantee the smoothness of the trajectory so that the mechanical structure has no vibrations added to a continuous curve that does not overshoot the final target. To satisfy kinematic and dynamic constraints, instead, we need to take care of some inequalities for velocity and acceleration, plus some other constraints related to the fact that $s(t)$ is like the percentage of the path completed at time t :

$$-\dot{s}_{max} \leq \dot{s}(t) \leq \dot{s}_{max}, \dot{s}_{max} > 0$$

$$-\ddot{s}_{max} \leq \ddot{s}(t) \leq \ddot{s}_{max}, \ddot{s}_{max} > 0, \ddot{s}_{max}^+ > 0$$

$$s(t_0) = 0, s(t_f) = 1$$

$$\dot{s}(t_0) = \dot{s}(t_f) = 0$$

$$\ddot{s}(t_0) = 0, \ddot{s}(t_{0+}) = \ddot{s}_{max}^+$$

$$\ddot{s}(t_f) = \ddot{s}_{max}^-, \ddot{s}(t_{f+}) = 0$$

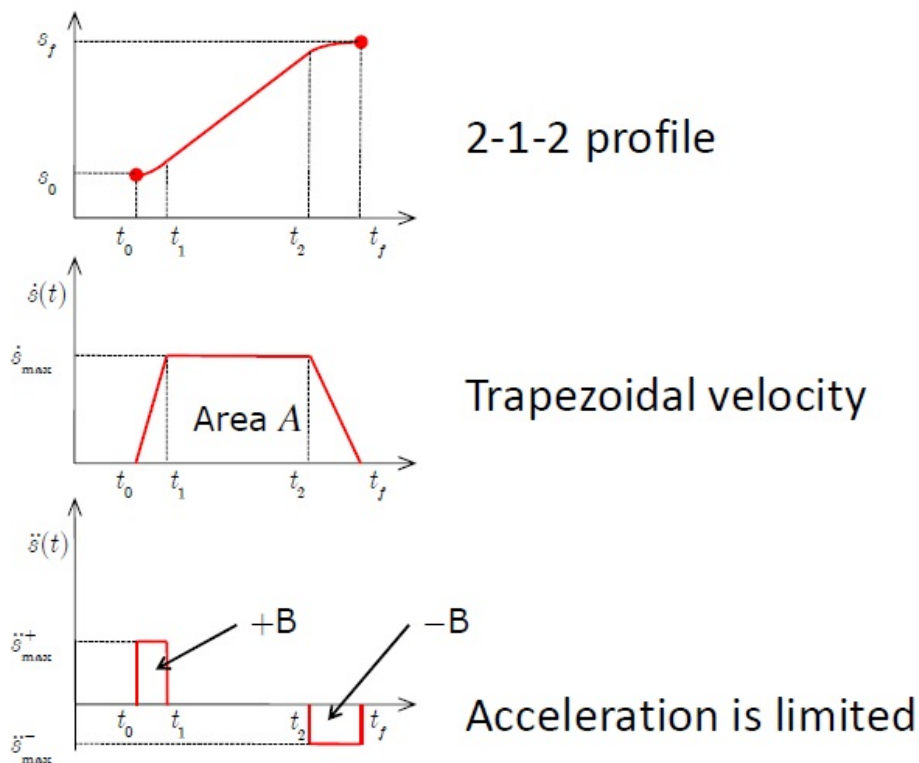


Figure 6: position, velocity and acceleration profiles

Since the position, velocity and acceleration assume three different behaviour during time, they can be described by different time laws, each one corresponding to a certain interval. We define:

$$L_1 = \{t : t_0 \leq t < t_1\} = [t_0, t_1)$$

$$L_2 = \{t : t_1 \leq t < t_2\} = [t_1, t_2)$$

$$L_3 = \{t : t_2 \leq t < t_f\} = [t_2, t_f)$$

So that we obtain:

$$\ddot{s}(t) = \begin{cases} \ddot{s}_{\max}^+ & t \in L_1 \\ 0 & t \in L_2 \\ -\ddot{s}_{\max}^- & t \in L_3 \end{cases}$$

$$\dot{s}(t) = \begin{cases} \ddot{s}_{\max}^+(t-t_0) + \dot{s}_0 & t \in L_1 \\ \dot{s}_{\max} & t \in L_2 \\ \dot{s}_{\max} - \ddot{s}_{\max}^-(t-t_2) & t \in L_3 \end{cases}$$

$$s(t) = \begin{cases} \frac{1}{2} \ddot{s}_{\max}^+(t-t_0)^2 + \dot{s}_0(t-t_0) + s_0 & t \in L_1 \\ \dot{s}_{\max}(t-t_1) + s_1 & t \in L_2 \\ -\frac{1}{2} \ddot{s}_{\max}^-(t-t_2)^2 + \dot{s}_{\max}(t-t_2) + s_2 & t \in L_3 \end{cases}$$

Imposing the continuity constraints between intervals it is possible to compute the partial percentages of space and some intervals in which the speed is constant or in which there is an acceleration or deceleration. However, there are cases in which it is not possible to reach the maximum velocity inside the trajectory, otherwise the joint will not be able to stop in the end point. This is what is called as ‘bang-bang trajectory’ in acceleration.

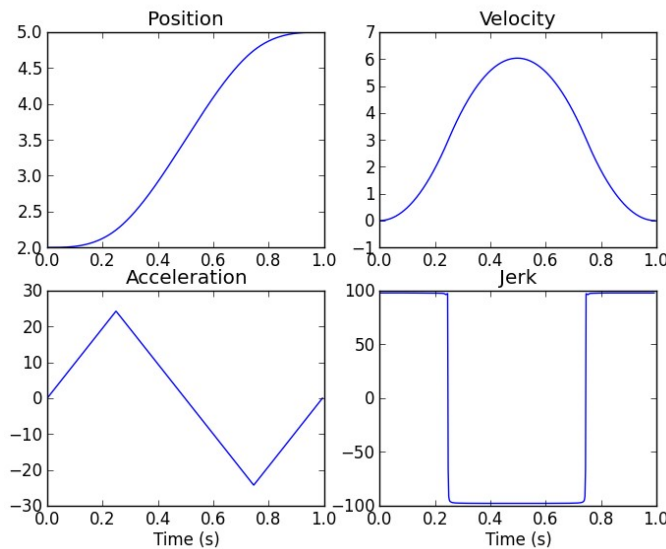


Figure 7: bang-bang trajectory profiles

In order to be managed by discrete controllers, continuous variables need to be sampled in intervals with a period T directly related to control specifications. In practice, since it is very difficult that the commutation times will coincide with the sampling ones, it is necessary to consider also additional constraints, particularly in order to avoid violating profile constraints. This means that computing recursively the new profile, acceleration must be limited so that velocity does not exceed its maximum value. The same must be valid also for the deceleration phase, since the zero final speed must be reached uniformly and without exceeding the maximum acceleration. This point the computation of both velocity and position is made using some interpolation algorithms, able to guarantee fast results obtaining just one of the two quantities and then find the other by approximation. In case of incremental interpolation from the velocity, obtained as said before, it is possible to apply the Euler formula for derivatives using a backward or a forward relation, which must be chosen according to the available data, obtaining respectively:

$$\dot{s}_k \approx (s_k - s_{k-1}) / T$$

$$\dot{s}_k \approx (s_{k+1} - s_k) / T$$

while with the absolute interpolation from the position we compute velocity as:

$$\dot{s}_k \approx (s_k - s_{k-1}) / T$$

In order to define the geometric path inside the joint space, it is possible or to define a vector of 6 dimensional joint values, from the first q_0 to the last q_f , or, much more rarely, through a parametric curve. This way the definition of the joint values becomes a series of stop and go motions between two consecutive q_i , so that the next is reached with zero velocity

1.1.6. TASK SPACE TRAJECTORY PLANNING

For the most used task trajectory planning, it is necessary to consider separately the position and the orientation variables, always taking care of their derivatives, particularly for what concerns the orientation angles. Also this time the position variables can be expressed through a vector or a parametric curve and then, in the first case all the adjacent points are linked together while in the second the parametric curve represents itself the motion law. For what concerns the orientation variables instead, there are three possibilities, both related to rotation matrices which must be kept orthonormal and with unitary determinant during the planning phase.

The first method is the axis-angle, and consists on the computation of the incremental rotation R_L , which links R_0 and R_f .

$$R_f = R_0 R_L$$

$$R_L = R_0^{-T} R_f$$

From it we obtain the total variation of the rotation angle and then the u axis around which the rotation has took place.

$$\Delta\vartheta = \pm \arccos \left(\frac{1}{2} (\text{tr } R_L - 1) \right)$$

$$S = \frac{1}{2} (R_L - R_L^T) / (\sin \Delta\vartheta)$$

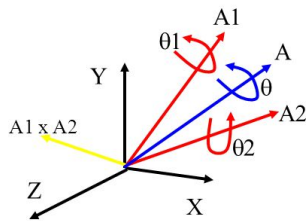
At the end, if we keep constant the u axis, we are able to plan the angle necessary to pass from the starting to the end orientation.

$$R_k = R(u, \vartheta_k)$$

$$\vartheta_k = s_k \Delta \vartheta$$

This method is simple to implement and gives good geometrical for the performed movement, as can be seen in the following example:

Axis-angle Interpolation



1. Interpolate axis from A1 to A2 Rotate axis about A1 x A2 to get A

2. Interpolate angle from theta1 to theta2 to get theta

3. Rotate the object by theta around A

Figure 8: axis-angle interpolation example

For the planar sliding method, instead, the motion is seen as the composition of two rotations, one around a fixed axis u obtained as the one orthogonal to the plane composed by k_0 and k_f , and the second around the moving local axis k .

$$R_k = R(u, \beta_k) R(k, \alpha_k)$$

Where

$$\beta = \arcsin \|k_0 \times k_f\|$$

$$u = (k_0 \times k_f) / \sin \beta$$

$$\alpha = \arcsin \|(R(u, \beta)^T j_0) \times j_f\|$$

$$\alpha_k = s_k \alpha$$

$$\beta_k = s_k \beta$$

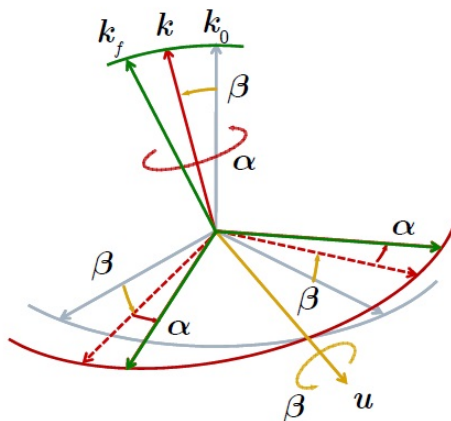


Figure 9: planar sliding interpolation

Finally the planning can be done easier using the three Euler angles, each of them planned according to the convex combination.

$$R_k = R(k, \phi_k) R(i, \vartheta_k) R(k, \psi_k)$$

$$\phi_k = \phi_0 + s_k(\phi_f - \phi_0)$$

$$\vartheta_k = \vartheta_0 + s_k(\vartheta_f - \vartheta_0)$$

$$\psi_k = \psi_0 + s_k(\psi_f - \psi_0)$$

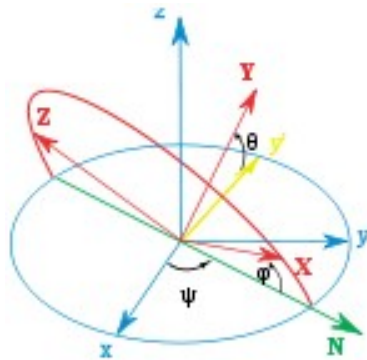


Figure 10: Euler angle interpolation

Once the task space trajectory planning has been computed, it is necessary to apply the inverse position KF in order to obtain the corresponding q_k . In case the sampling period results too small to compute it, it is possible to use the micro-macro interpolation. This way the inverse KF is computed only on multiples of the sampling period and then the consecutive vectors are interpolated linearly to obtain approximated values.

1.2. ANALYSIS OF THE STATE OF ART OF PROGRAMMING LANGUAGES FOR COMAU, FANUC AND ABB

Many researchers studied, during the years, multiple types of interpolation methods, in order to allow the industries to adapt the robots to their specific needs. Beyond all of them, in 2014 Menasri, Oulhadj, Daachi, Nakib and Siarry from University Paris-Est Creteil, concentrated their attention also on the avoidance of abrupt movements, ensuring this way at the same time the smoothness of the trajectory and the velocity and acceleration levels using position functions at least two times differentiable. The starting point of this procedure is based on the definition of a sets of points called knots by which it is mandatory for the trajectory to pass through. So the problem of trajectory planning becomes a problem of interpolation.

For the interpolation purposes it is possible to use many types of techniques, divided in polynomial functions as cubic splines or B-splines and trigonometric functions. The main difference between the two is related to the fact that in case of variation of the available data polynomial functions are easier to be modified but they need to be at least of fourth order to guarantee the smoothness of the trajectory also at the acceleration level, while this is always verified by using the trigonometric functions.

Using polynomial functions, this problem can be solved using a “classical” approach or an heuristic one, based on the genetic algorithm where each possible solution of the problem is coded as a string called chromosome. Then, through crossover and mutation, some changes can be randomly introduced inside the chromosome adding also the recombination between some of them until new improving solutions are reached. However, this type of implementation, can require much more time than a polynomial one, due to the fact that there is no a clear recipe about the most improving crossovers and mutation, so at the end it could be the case that a new improving solution is not found at all.

If we consider instead of a purely theoretical research a much more practical approach, from the literature we can say that there are basically three ways to program a robot:

- Teach pendant: it is the most popular method (more than 90% of the robots are programmed through it, according to the British Automation and robot Association). To program the robot, the operator moves it from point-to-point, using the buttons on the pendant to move it around and save each position individually so that it can be used within the robot program. The coordinate systems available on standard jointed arm robots are:
 - the joint coordinated: each joint is independent and can be moved in every direction;
 - the global coordinates: they are the coordinates X, Y and Z of the robot global axis system and can be used to perform both rotations and translations to move the tool centre point in a specific position;
 - the tool coordinates: the coordinates of the tool centre point;
 - the workpiece coordinates: they are useful when it is necessary to make small adjustments to the program and they can be easily done along a major axis of the coordinate system than along a general line. This can be done when on the robot it is possible to set up a coordinate system wherever inside the working area and produces a result which is very similar to the one obtained through the position and orientation of the global coordinate system

When the whole program has been learned, the robot can play back the points at full speed. This method is very simple to be implemented, and can be used in all that situations in which simple movements are required. The main disadvantage is related to the fact that every time a modification is necessary, it can take quite a long time during which the robot is of course out of production and this is why it is convenient to use the teach pendant only when the robot has to do the same movements for all its life.

- Lead trough: the robot controller records joint positions at a fixed time interval while it is moved physically by the operator. The main disadvantage of this type of programming is related both to the dimensions of the robot, due to the fact that if they are too big they cannot be easily moved by the operator, and to the difficulties related to reprogramming the robot in the case in which hesitations or inaccuracies are introduced inside the program. This is why this solution now is almost disappeared, a part some cases where it is still used for paint spraying applications.
- Off-line programming: the CAD models of the components and their models are used to program the robot. The main difference between the others teach programming is related to the fact that intelligent tools are here available so CAD data can be used to

generate sequences of location and process information. Even if this method is not so common it has many benefits, as reducing time for programming because it is easier and assist cell design to allow process optimization.

1.2.1. COMAU TRAJECTORY PLANNING

Trajectory planning by Comau is possible through the remote control performed by Teach Pendant. On December 2017, Comau researchers realized a new application called PickAPP in order to make the programming of industrial robots more intuitive, faster and so easier. This way it is possible to move a robot leading it to perform pick and place issues just defining the points of its trajectory without the necessity to learn a new programming language. Moreover, due to the fact that the application works directly on the robot's joints, some particular trajectories or gripper's actions can be easily implemented.

Programming language used by Comau is called *PDL2* and it appears very similar to Pascal, since they both provide a structured programming strategy. Between all the available features, the most important are related to the fact that there are predefined identifiers of variables, constants and procedures which can be used for robotic applications, as well as a large number of instructions in order to allow the gripper's movements. Moreover, it is also possible to control some events while the program is in execution. All the instructions which compose the code are compiled and translated in object code, and finally saved inside the memory. It is also possible to program the robot with a direct approach through *EZ*, which memorizes all the movements and then translate them into *PDL2*. This approach is simpler because much more direct, but allows only to realize simple programs.

The basic program is started by the key word PROGRAM and finished with END. In between it is possible to initialize some specific position as VAR POSITION so that, when the execution begins after the BEGIN CYCLE instruction, all the movements can be performed just selecting the proper position and then telling the robot to use the gripper.

The most important variables are the VECTOR and the POSITION. The first is composed of three elements, X, Y and Z of real type, while the second is composed by three position coordinates (X, Y and Z), three Euler angles (E1, E2 and E3) of real type and a component of type string. This structure allows to describe the position and orientation of a triad with respect to another which is used as a reference. Joint variables are instead contained inside the JOINTPOS structure, for non redundant arms.

To control the movements, instead, the main function is called MOVE TO, which allows the motion from the starting point to a specified target. Moreover, it is possible to decide the type of trajectory. For example MOVE JOINT TO is used for the joint space and represents the default configuration in case nothing else is specified, MOVE LINEAR TO is for the task space and MOVE CIRCULAR TO is used to describe a circumference arc.

1.2.2. FANUC PROGRAMMING LANGUAGE

The virtual environment offered by Fanuc is called RobotGuide. It allows to create a layout of both robot and workpiece thanks to a library of objects and then check the robot posture by graphic jog. Moreover CAD data can be imported to create parts by the modelling function. This way users can create the actual program from the shapes designed inside the graphic screen and from the instruction given to a virtual teach pendant because a code is

automatically generated so that, once the simulation has given the desired results, it can be directly downloaded inside the robot to see it moving.

It is also possible to add some specific software for example to simulate a robotic process or study feasibility options for 3D applications, for painting or pallet issues, etc..

2.1.1. ABB PROGRAMMING LANGUAGE

ABB's programming environment is called RobotStudio. In the task space the robot can be moved in many different ways, depending on the situation. This to guarantee the greatest adaptability to all the situations.

The most general sequence which is recommended is composed by these steps:

- First of all it is necessary to define the path we want the robot to follow and then its target. This can be done creating a curve of the desired shape until the target is reached, or just creating the targets. The curve can be straight, a circumference described both through the center and the radius or by three points, an ellipse, a spline, etc.
- Once the targets are defined, it is necessary to control their orientation to be sure that the trajectory followed by the robot is the most efficient and passes through all the desired targets.
- Generate the RAPID code from RobotStudio objects to obtain the program simulation. If it is necessary to modify something, it is possible to do it by editing the program through text.
- Finally, it is necessary to verify that the robot and its TCP move sufficiently far from the utilities around them to avoid collisions by simulating the program.

Also in this case the code is automatically generated by the graphic environment, so it is not necessary to learn a new programming language because everything can be done in a virtual situation before being downloaded inside the robot for the real application.

2. DEFINITION OF THE SYSTEM

The aim of my work is, starting from a predefined industrial environment, to program three ABB robots to perform pick and place tasks, trying to optimize them in terms of cycle time and of trajectories.

In this specific case, we are discussing about the presence of three anthropomorphic robots, with 6 degrees of freedom able to guarantee the maximum possible dexterity used to pick and place objects of different types and weights. Articulated or anthropomorphic means that the shoulder has three revolute joints, the first one vertical and the other two horizontal and parallel. The structure is very similar to human body, with trunk, arm and forearm with a final wrist so it provides the best dexterity and this is why it is used in most of the industrial applications. Task space is a sort of sphere sector, even if there is no a direct correspondence between joint and Cartesian coordinates, but accuracy is not constant inside the task space.

Particularly two of them are equal with a maximum capacity of 60 kg and are used to pick and place clutch's mechanisms of a maximum weight of 8 kg, with a gripper of about 32 kg. The third one, instead, is used to move rivets of few grams, so it is sufficient with a maximum capacity of 5 kg. They both provide a double gripper to optimize the cycle time in order to be able to match the specification of 15s.

The system scenario is composed by 13 stations, positioned in space in a pre-determined way, through which vehicular clutches are balanced, tested and finally marked.

The line behaves as follows:

- The clutch, from the external line, is moved to STA, where it is picked by robot R1 with gripper 1 and moved to ST5. If in ST5 there is already an item, it is picked up by gripper 2.
- Manipulator M2 does one step to move the item on ST1 for the measurement of the residual imbalance. Then it moves again of 72° , so the clutch arrives in ST2.
- Robot R2 picks the requested rivets from the rails of the rivet feeder (from 1 to 4, the first always with gripper1) and place them on the clutch present in ST2. Since the station is able to rotate, R2 performs the task always in the same position.
- Clutch is moved to both ST3 and ST4 to complete the riveting, then it is moved to ST5.
- Item is picked up from ST5 by R1 and moved alternatively to ST6 and ST7.
- Once M2 has completed the fingers correction on the clutch, it is picked up by robot R3 (alternatively from ST6 and ST7) and placed in ST8.
- Finally, when M3 has finished the functional control, the mechanism is moved to ST9 and it will be marked. Once the rotating table has done 3 steps, in ST12 line is discharged by the last manipulator.

NOK conveyors are finally used to move away damaged parts.

The whole line is organized as follows:

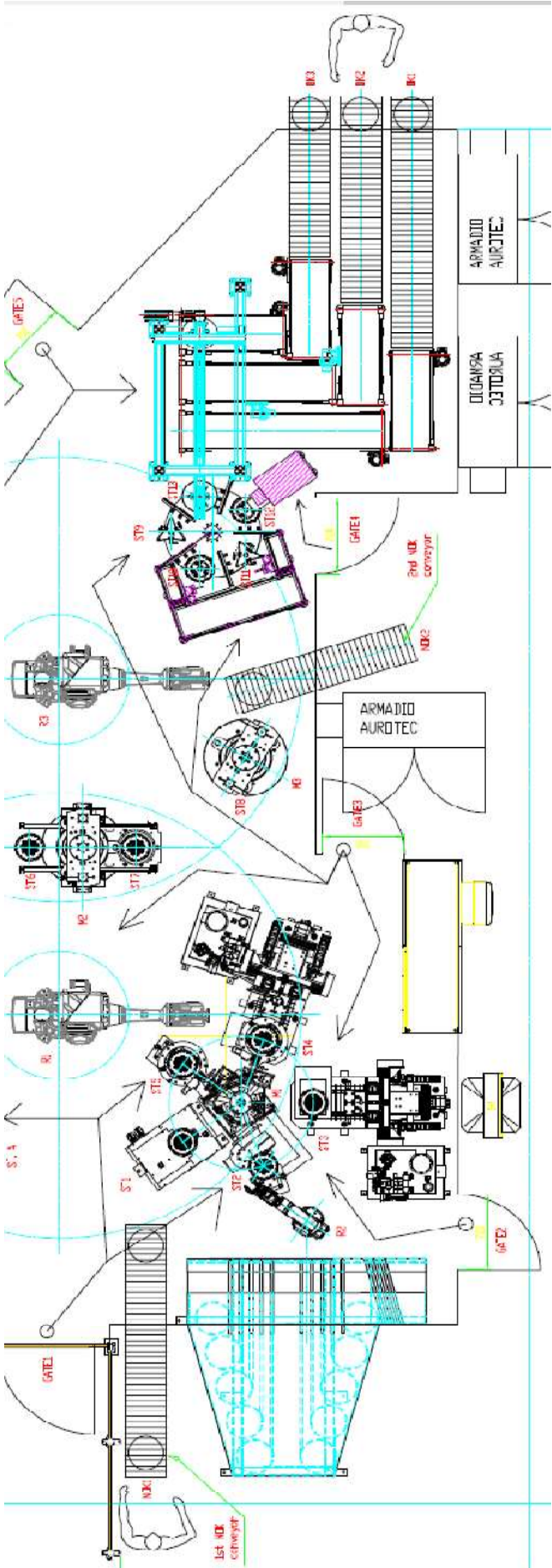


Figure 11: line layout

2.1 ROBOT R1

The production flow starts from the left side of the line, where we have the first robot, R1 for the representation. It is an IRB 4600 by ABB, with 60 kg of payload and a possible extension of the arm up to 2.05 m. It is equipped with a double gripper with three fingers each, able to pick up the clutches. Moreover it communicates with a PLC Siemens 1500, which works as a master of the whole line, through the Profinet bus. Its task is, depending on the gripper, to pick and place the clutches. Particularly the one called gripper 1 (on the up side from the robot point of view) performs its task from the entry of the line STA to ST5, while for what concerns gripper 2, it takes the items from ST5 and move them alternatively in ST6 and ST7. Moreover, if clutches are defined NOK, so some faults are detected during the process, they are moved from STA to NOK1_A by gripper 1 in case of problems caused by assembly task, while they are moved from ST5 to NOK1_B with gripper 2 if balancing has not been done correctly.

Below the detailed picture related to R1 task:

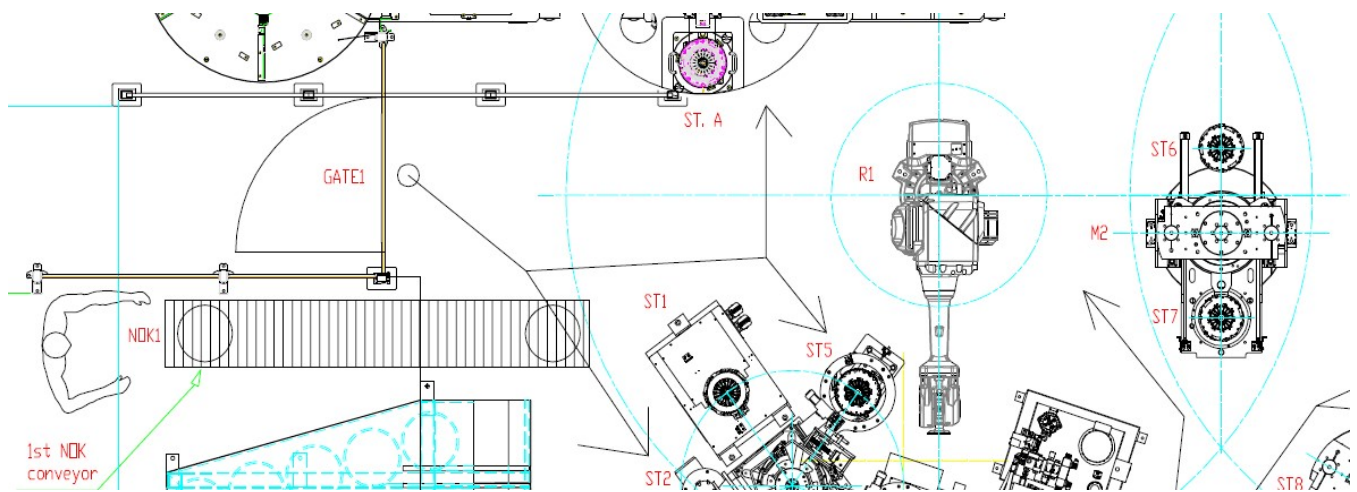


Figure 12: R1 task description

Due to the fact that there are 2 grippers, for programming clarity it was necessary to define them as two different tools, so that also the pick and place procedure could be written in a specific way for each of them.

```
TASK PERS tooldata tGripper1:=[TRUE, [[-355, -35, 175], [1, 0, 0, 0]], [31.5, [-6.4, -1.9, 64.7], [1, 0, 0, 0], 0, 2.747, 3.586]];
TASK PERS tooldata tGripper2:=[TRUE, [[355, 35, 175], [1, 0, 0, 0]], [31.5, [-6.4, -1.9, 64.7], [1, 0, 0, 0], 0, 2.747, 3.586]];
```

The whole gripper is symmetric with respect to its center, which coincides with the robot's TCP. This is why in practice the center of each gripper has in module the same position, while the direction changes. In fact, the tool coordinate system is rotated of 90° around y axis with respect to the base one. So, for gripper 1, x and y directions are negative (it is the upper one), while for gripper 2 they are positive. Since z direction is coming outside the plane, it is always positive.

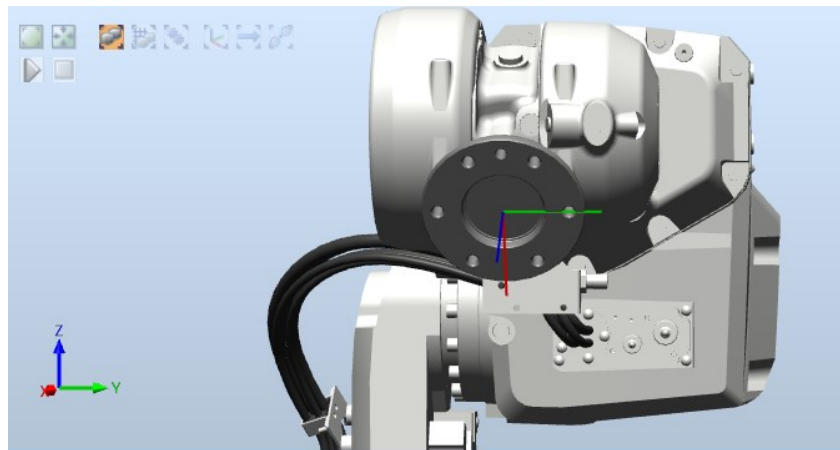


Figure 13: TCP coordinate system with respect to the base one

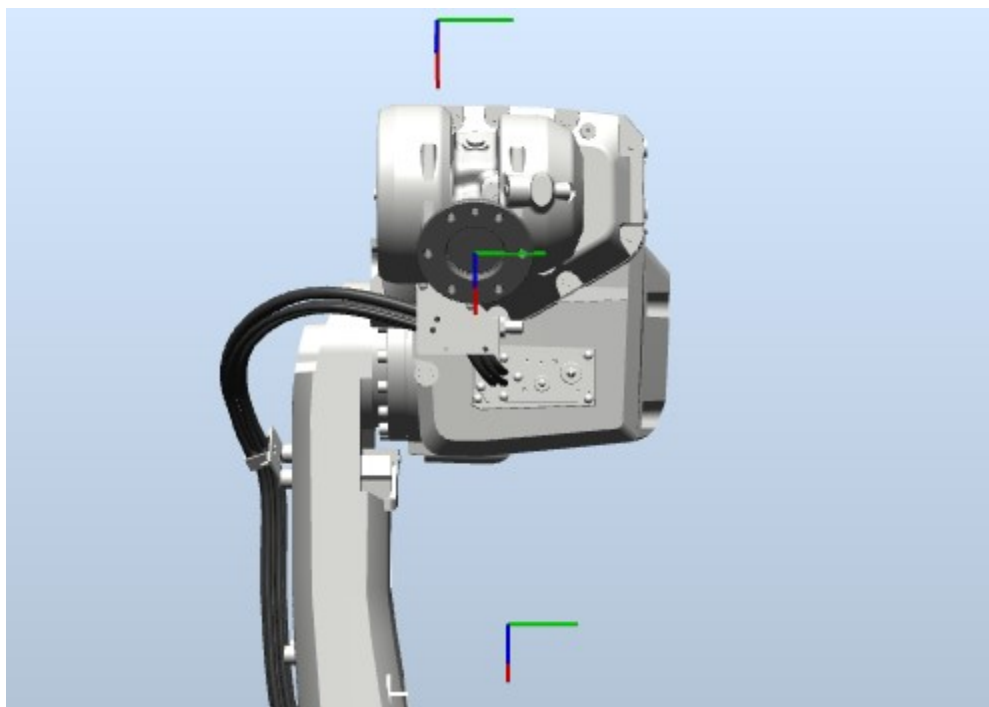


Figure 14: gripper 1 and 2 reference frames with respect to the TCP

2.1.1. I/O CONFIGURATION

To properly build up the I/O system of R1 robot, I added the Profinet option inside the configuration section of RobotStudio, reserving 16 bytes of Digital Inputs and 16 bytes of Digital Outputs. All those signals are exchanged during the cycle with the PLC, particularly all the inputs are commands from the PLC to the robot related to the system state or to the movement it can do, while the outputs represent the confirmation that the commands have been received or that the movements have been executed.

The screenshot shows the 'Configurazione' window in RobotStudio. On the left, the 'I/O System' tree is expanded, showing 'PROFINET' and 'PN_Internal_Device' with '02: DI_16_bytes' selected. The main table lists 63 digital input signals, each mapped to a specific DI_16_bytes address. The signals are categorized by their function, such as system control, emergency, and gripper status.

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Acc
Drive_on	DI_16_bytes	Digital Input	0		byte0	Defe
Start	DI_16_bytes	Digital Input	1		byte0	Defe
Stop_program	DI_16_bytes	Digital Input	2		byte0	Defe
Soft_stop	DI_16_bytes	Digital Input	3		byte0	Defe
Quick_stop	DI_16_bytes	Digital Input	4		byte0	Defe
Start_main	DI_16_bytes	Digital Input	5		byte0	Defe
Reset_emergency_stop	DI_16_bytes	Digital Input	6		byte0	Defe
Reset_error_signal	DI_16_bytes	Digital Input	7		byte0	Defe
Stop_end_of_cycle	DI_16_bytes	Digital Input	8		byte1	Defe
Stop_end_of_instruction	DI_16_bytes	Digital Input	9		byte1	Defe
System_restart	DI_16_bytes	Digital Input	10		byte1	Defe
PP_main	DI_16_bytes	Digital Input	11		byte1	Defe
Emergency_off	DI_16_bytes	Digital Input	24		byte3	Defe
Auto_mode	DI_16_bytes	Digital Input	25		byte3	Defe
Home_position_request	DI_16_bytes	Digital Input	26		byte3	Defe
Pick_entry_line_G1	DI_16_bytes	Digital Input	32		byte4	Defe
Place_S05_G1	DI_16_bytes	Digital Input	33		byte4	Defe
Pick_S05_G2	DI_16_bytes	Digital Input	34		byte4	Defe
Place_S06_G2	DI_16_bytes	Digital Input	35		byte4	Defe
Place_S07_G2	DI_16_bytes	Digital Input	36		byte4	Defe
Place_NOK1_A_G1	DI_16_bytes	Digital Input	37		byte4	Defe
Place_NOK1_B_G2	DI_16_bytes	Digital Input	38		byte4	Defe
Pick_NOK_fromS05toNOK1_A_G2	DI_16_bytes	Digital Input	40		byte5	Defe
Pick_NOK_fromS05toNOK1_B_G2	DI_16_bytes	Digital Input	41		byte5	Defe
Pick_OK_S05_G2	DI_16_bytes	Digital Input	42		byte5	Defe
Pick_OK_TOX_G1	DI_16_bytes	Digital Input	43		byte5	Defe
Pick_NOK_TOX_G1	DI_16_bytes	Digital Input	44		byte5	Defe
Open_gripper1	DI_16_bytes	Digital Input	48		byte6	Defe
Clutch_in_gripper1	DI_16_bytes	Digital Input	49		byte6	Defe
Open_gripper2	DI_16_bytes	Digital Input	50		byte6	Defe
Clutch_in_gripper2	DI_16_bytes	Digital Input	51		byte6	Defe
Bit0_reference	DI_16_bytes	Digital Input	56		byte7	Defe
Bit1_reference	DI_16_bytes	Digital Input	57		byte7	Defe
Bit2_reference	DI_16_bytes	Digital Input	58		byte7	Defe
Bit3_reference	DI_16_bytes	Digital Input	59		byte7	Defe
Bit4_reference	DI_16_bytes	Digital Input	60		byte7	Defe
Bit5_reference	DI_16_bytes	Digital Input	61		byte7	Defe
Bit6_reference	DI_16_bytes	Digital Input	62		byte7	Defe
Bit7_reference	DI_16_bytes	Digital Input	63		byte7	Defe
	DI_16_bytes	Digital Input				Defe

Figure 15: Inputs configuration

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Acc
Drive_on_ok	DO_16_bytes	Digital Output	0	byte0	Defe	^
Cycle_running	DO_16_bytes	Digital Output	1	byte0	Defe	
Emergency_stop	DO_16_bytes	Digital Output	2	byte0	Defe	
Error_signal	DO_16_bytes	Digital Output	3	byte0	Defe	
Robot_in_home_pos	DO_16_bytes	Digital Output	26	byte3	Defe	
Clutch_picked_form_entry_line	DO_16_bytes	Digital Output	32	byte4	Defe	
Clutch_placed_in_S05	DO_16_bytes	Digital Output	33	byte4	Defe	
Clutch_picked_from_S05	DO_16_bytes	Digital Output	34	byte4	Defe	
Clutch_placed_in_S06	DO_16_bytes	Digital Output	35	byte4	Defe	
Clutch_placed_in_S07	DO_16_bytes	Digital Output	36	byte4	Defe	
Clutch_placed_in_NOK1_A	DO_16_bytes	Digital Output	37	byte4	Defe	
Clutch_placed_in_NOK1_B	DO_16_bytes	Digital Output	38	byte4	Defe	
Out_of_area_entry_line	DO_16_bytes	Digital Output	40	byte5	Defe	
Out_of_area_S05	DO_16_bytes	Digital Output	41	byte5	Defe	
Out_of_area_NOK_conveyor	DO_16_bytes	Digital Output	42	byte5	Defe	
Out_of_area_S06	DO_16_bytes	Digital Output	43	byte5	Defe	
Out_of_area_S07	DO_16_bytes	Digital Output	44	byte5	Defe	
Gripper1_opened	DO_16_bytes	Digital Output	48	byte6	Defe	
Gripper1_closed	DO_16_bytes	Digital Output	49	byte6	Defe	
Gripper2_opened	DO_16_bytes	Digital Output	50	byte6	Defe	
Bit4_clutch_reference	DO_16_bytes	Digital Output	60	byte7	Defe	
Bit5_clutch_reference	DO_16_bytes	Digital Output	61	byte7	Defe	
Bit6_clutch_reference	DO_16_bytes	Digital Output	62	byte7	Defe	
Bit7_clutch_reference	DO_16_bytes	Digital Output	63	byte7	Defe	
	DO_16_bytes	Digital Output			Defe	

Figure 16: Outputs configuration

For what concerns the system signal, there are different types of stop to be able to differentiate situations. Particularly the *stop_program* is the flex pendant signal, from which brakes are immediately activated, while *quick_stop* and *soft_stop* interrupt the program execution braking the motors with different deceleration ramps. Moreover, the difference between the two is related to the fact that when *quick_stop* is activated, all the motors are deactivated for security reasons, viceversa this does not happen when *soft_stop* signal becomes high. *Stop_end_of_cycle*, instead, can be activated just if both the grippers are empty, while *stop_end_of_instruction* is just related to the end of a line of code or movement.

Since many types of clutches can be tested inside the line, it was necessary to take into account the fact that each one has its own dimensions and height. This particularly will interfere with the place position of the rivets, which can change both along the radial position and the vertical one, but can be important in case also the position of the mechanical plugs changes. To be able to manage directly all these mechanical information, the variable *reference* was created as a number which can assume all the values between 0 and 127, each corresponding to an alphanumeric code translated by the PLC. This way, once the reference is defined and passed to the robot, it is able to perform the tasks without the need of other information. The only negative aspect is related to the fact that the WorkObjects must be defined in practice for each reference and this can take quite a long time, depending on the number of possible references.

Moreover, the same reference bits that are received as inputs are also set as output during the program execution, so the correctness of the signals exchanged can be easily checked.

2.2 ROBOT R2

Proceeding through the flow we have the second robot, R2 for the representation. It is an IRB 1200 by ABB, with 5 kg of payload and a possible extension of the arm up to 0.9 m. It is equipped with a double gripper, each one composed by a vacuum switch able to pick up rivets of different sizes and weights. Moreover it communicates with a PLC Siemens 1500, which works as a master of the whole line, through the Profinet bus. Its task is, given the rivet code, to pick and place it on the clutch cover which is present on ST2.

The detailed picture related to R2 is the following one:

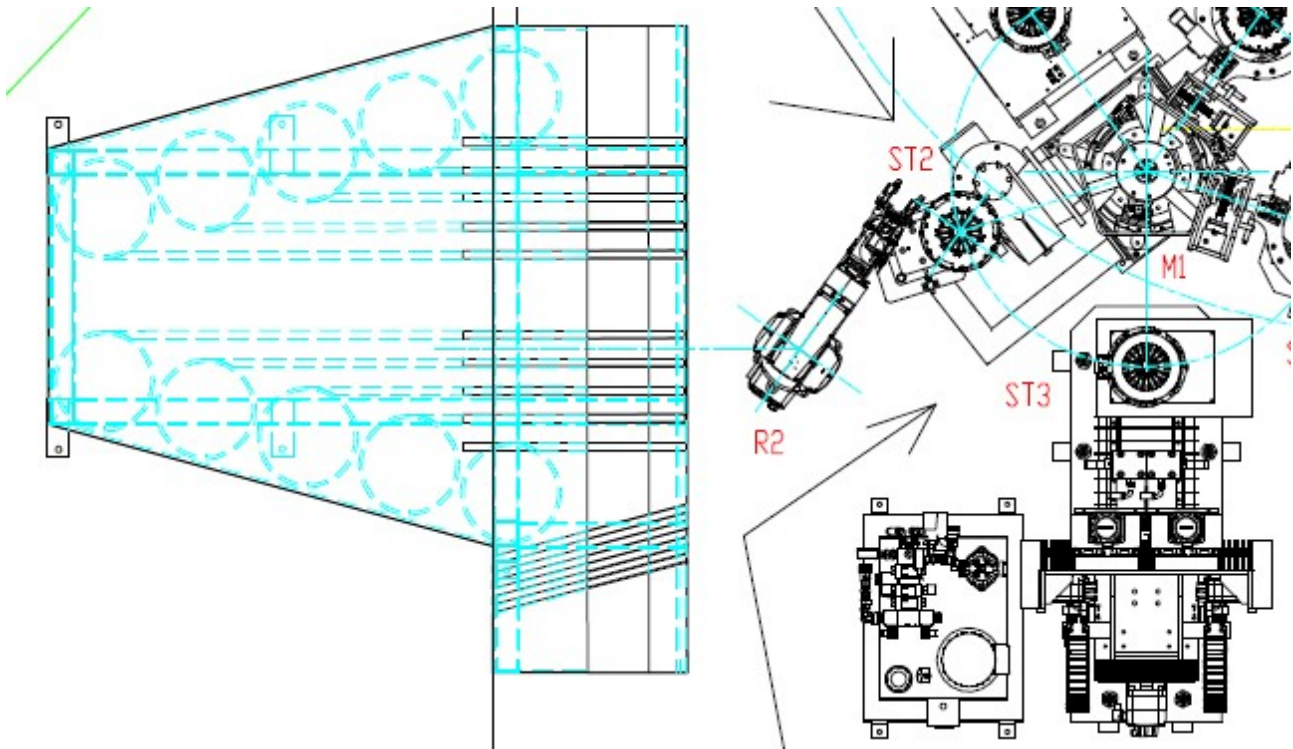


Figure 17: R2 task description

From that it is clear that the two WorkObjects that interact with R2 are the rivet feeder splitted in 14 positions for the pick task and the clutch cover for the place one.

```
TASK PERS tooldata tGripper1=[TRUE,[[ -102.537,12.3783,99.1169],[1,0,0,0]],[2.7,[-53.1,-18.7,-7.2],[1,0,0,0],0.01,0.01,0.007]];
TASK PERS tooldata tGripper2=[TRUE,[[ -76.1507,-71.7797,99.3901],[1,0,0,0]],[2.7,[-53.3,-18.8,-9.7],[1,0,0,0],0.012,0.012,0.007]];
```

From the coordinates of the tools, it is evident that this time they are not symmetric anymore, but placed in a configuration which appears rotated toward right from the robot point of view.

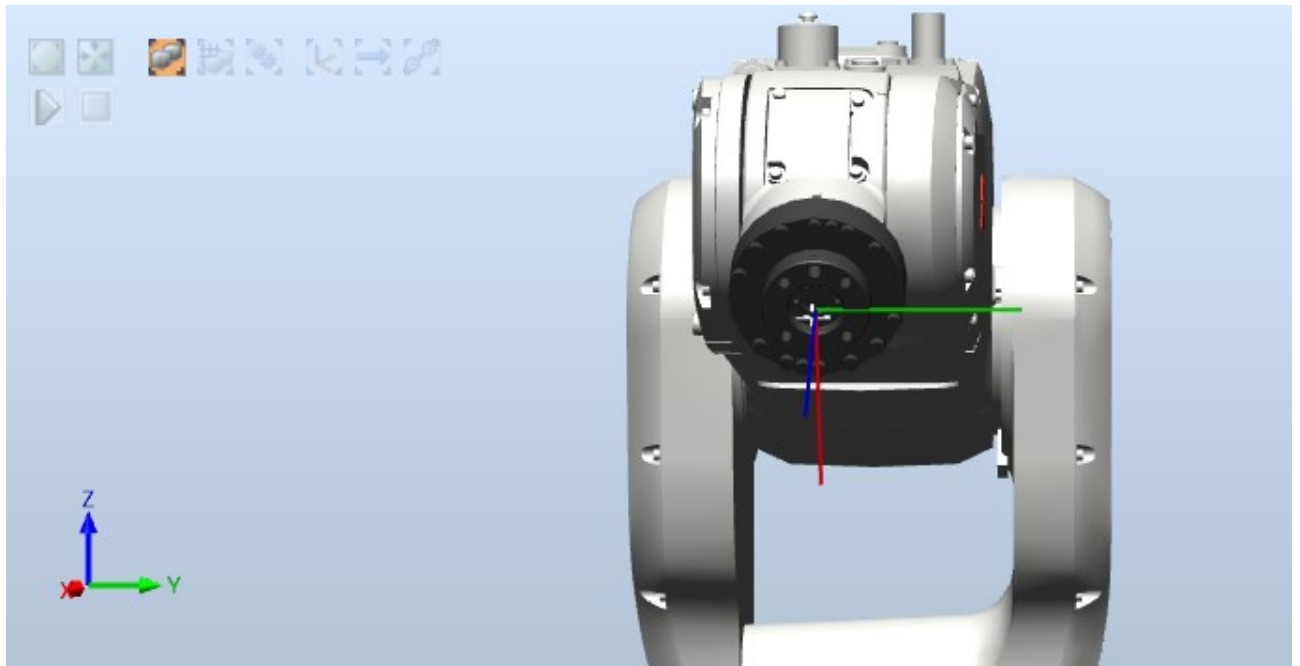


Figure 18: TCP reference frame with respect to the base one

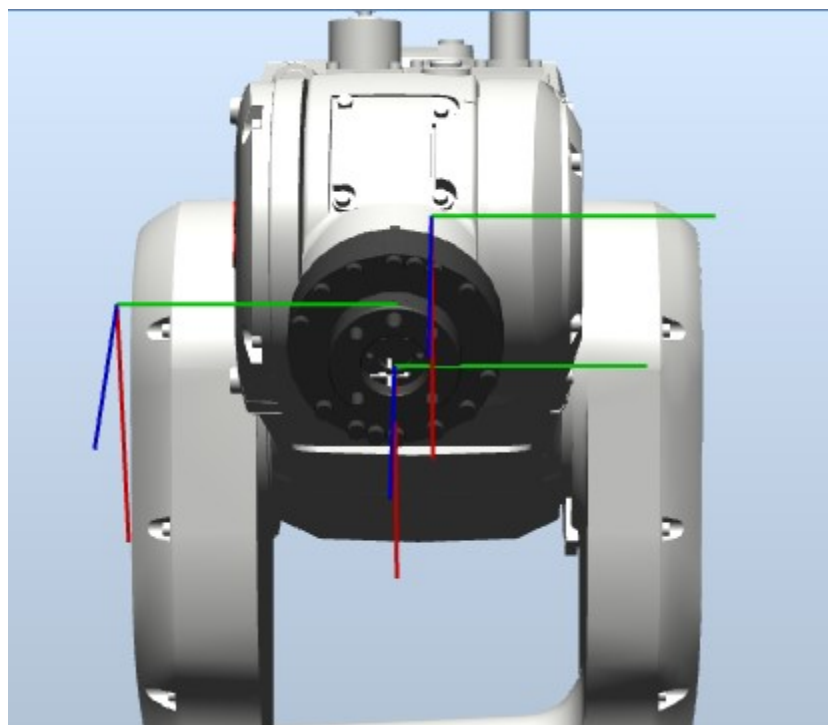


Figure 19: gripper 1 and 2 reference frame with respect to the TCP

2.2.1. I/O CONFIGURATION

To properly build up the I/O system of R2 robot, I added the Profinet option inside the configuration section of RobotStudio, reserving 16 bytes of Digital Inputs and 16 bytes of Digital Outputs, as I did also for R1. All those signals are exchanged during the cycle with the PLC, particularly all the inputs are commands from the PLC to the robot related to the system state or to the movement it can do, while the outputs represent the confirmation that the commands have been received or that the movements have been executed.

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Access Level
Drive_on	DI_16_bytes	Digital Input	0		byte0	Default
Start	DI_16_bytes	Digital Input	1		byte0	Default
Stop_program	DI_16_bytes	Digital Input	2		byte0	Default
Soft_stop	DI_16_bytes	Digital Input	3		byte0	Default
Quick_stop	DI_16_bytes	Digital Input	4		byte0	Default
Start_main	DI_16_bytes	Digital Input	5		byte0	Default
Reset_emergency_stop	DI_16_bytes	Digital Input	6		byte0	Default
Reset_error_signal	DI_16_bytes	Digital Input	7		byte0	Default
Stop_end_of_cycle	DI_16_bytes	Digital Input	8		byte1	Default
Stop_end_of_instruction	DI_16_bytes	Digital Input	9		byte1	Default
System_restart	DI_16_bytes	Digital Input	10		byte1	Default
PP_main	DI_16_bytes	Digital Input	11		byte1	Default
Emergency_off	DI_16_bytes	Digital Input	24		byte3	Default
Auto_mode	DI_16_bytes	Digital Input	25		byte3	Default
Home_position_request	DI_16_bytes	Digital Input	26		byte3	Default
Place_rivet1	DI_16_bytes	Digital Input	32		byte4	Default
Place_rivet2	DI_16_bytes	Digital Input	33		byte4	Default
Pick_rivet1	DI_16_bytes	Digital Input	34		byte4	Default
Pick_rivet2	DI_16_bytes	Digital Input	35		byte4	Default
Rivet1_in_gripper1	DI_16_bytes	Digital Input	48		byte6	Default
Rivet2_in_gripper2	DI_16_bytes	Digital Input	49		byte6	Default
Bit0_reference	DI_16_bytes	Digital Input	56		byte7	Default
Bit1_reference	DI_16_bytes	Digital Input	57		byte7	Default
Bit2_reference	DI_16_bytes	Digital Input	58		byte7	Default
Bit3_reference	DI_16_bytes	Digital Input	59		byte7	Default
Bit4_reference	DI_16_bytes	Digital Input	60		byte7	Default
Bit5_reference	DI_16_bytes	Digital Input	61		byte7	Default
Bit6_reference	DI_16_bytes	Digital Input	62		byte7	Default
Bit7_reference	DI_16_bytes	Digital Input	63		byte7	Default
Bit0_rivet1	DI_16_bytes	Digital Input	72		byte9	Default
Bit1_rivet1	DI_16_bytes	Digital Input	73		byte9	Default
Bit2_rivet1	DI_16_bytes	Digital Input	74		byte9	Default
Bit3_rivet1	DI_16_bytes	Digital Input	75		byte9	Default
Bit0_rivet2	DI_16_bytes	Digital Input	88		byte11	Default
Bit1_rivet2	DI_16_bytes	Digital Input	89		byte11	Default
Bit2_rivet2	DI_16_bytes	Digital Input	90		byte11	Default
Bit3_rivet2	DI_16_bytes	Digital Input	91		byte11	Default
	DI_16_bytes	Digital Input				Default

Figure 20: Inputs configuration

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Access
Drive_on_ok	DO_16_bytes	Digital Output	0		byte0	Default
Cycle_running	DO_16_bytes	Digital Output	1		byte0	Default
Emergency_stop	DO_16_bytes	Digital Output	2		byte0	Default
Error_signal	DO_16_bytes	Digital Output	3		byte0	Default
Robot_in_home_pos	DO_16_bytes	Digital Output	26		byte3	Default
Rivet1_placed	DO_16_bytes	Digital Output	32		byte4	Default
Rivet2_placed	DO_16_bytes	Digital Output	33		byte4	Default
Rivet1_picked	DO_16_bytes	Digital Output	34		byte4	Default
Rivet2_picked	DO_16_bytes	Digital Output	35		byte4	Default
Out_of_ST2	DO_16_bytes	Digital Output	36		byte5	Default
Out_of_rivet_feeder	DO_16_bytes	Digital Output	37		byte5	Default
Vacuum_activation_on_gripper1	DO_16_bytes	Digital Output	48		byte6	Default
Vacuum_activation_on_gripper2	DO_16_bytes	Digital Output	49		byte6	Default
Bit0_clutch_reference	DO_16_bytes	Digital Output	56		byte7	Default
Bit1_clutch_reference	DO_16_bytes	Digital Output	57		byte7	Default
Bit2_clutch_reference	DO_16_bytes	Digital Output	58		byte7	Default
Bit3_clutch_reference	DO_16_bytes	Digital Output	59		byte7	Default
Bit4_clutch_reference	DO_16_bytes	Digital Output	60		byte7	Default
Bit5_clutch_reference	DO_16_bytes	Digital Output	61		byte7	Default
Bit6_clutch_reference	DO_16_bytes	Digital Output	62		byte7	Default
Bit7_clutch_reference	DO_16_bytes	Digital Output	63		byte7	Default
Bit0_gripper1	DO_16_bytes	Digital Output	72		byte9	Default
Bit1_gripper1	DO_16_bytes	Digital Output	73		byte9	Default
Bit2_gripper1	DO_16_bytes	Digital Output	74		byte9	Default
Bit3_gripper1	DO_16_bytes	Digital Output	75		byte9	Default
Bit0_gripper2	DO_16_bytes	Digital Output	88		byte11	Default
Bit1_gripper2	DO_16_bytes	Digital Output	89		byte11	Default
Bit2_gripper2	DO_16_bytes	Digital Output	90		byte11	Default
Bit3_gripper2	DO_16_bytes	Digital Output	91		byte11	Default
	DO_16_bytes	Digital Output				Default

Figure 21: Outputs configuration

For what concerns the system signal, there are different types of stop to be able to differentiate situations. Particularly the *stop_program* is the flex pendant signal, from which brakes are immediately activated, while *quick_stop* and *soft_stop* interrupt the program execution braking the motors with different deceleration ramps. Moreover, the difference between the two is related to the fact that when *quick_stop* is activated, all the motors are deactivated for security reasons, viceversa this does not happen when *soft_stop* signal becomes high. *Stop_end_of_cycle*, instead, can be activated just if both the grippers are empty, while *stop_end_of_instruction* is just related to the end of a line of code or movement.

Since many types of clutches can be tested inside the line, it was necessary to take into account the fact that each one has its own dimensions and height. This particularly interferes with the place position of the rivets, which can change both along the radial position and the vertical one. All these information are contained inside the parameter *reference*, a number which can assume all the values between 0 and 127, each corresponding to an alphanumeric code translated by the PLC.

The same considerations can be done also for the rivets: depending on the PLC request, up to 14 different types of rivet, varying in weight, can be chosen to balance the clutch. For this purpose, it was necessary to create an array of positions so that, each time a new rivet is requested to be picked up, the pick position is updated.

Moreover, for both reference and rivet code, the same bits that are received as inputs are also set as output during the program execution, so the correctness of the signals exchanged can be easily checked.

2.3 ROBOT R3

Finally we have the third robot, R3 for the representation. It is again an IRB 4600 by ABB, with 60 kg of payload and a possible extension of the arm up to 2.05 m. It is equipped with a double gripper with three fingers each, able to pick up the clutches. Moreover it communicates with a PLC Siemens 1500, which works as a master of the whole line, through the Profinet bus. Its task is, depending on the gripper, to pick and place the clutches. This time the one called gripper 2 (on the right side from the robot point of view) performs its task from ST6 or ST7 alternatively to ST8, while for what concerns gripper 1, it takes the items from ST8 and move them in ST9. Moreover, if clutches are defined NOK, so some faults are detected during the process, they are moved by gripper 2 to NOK_FC in case of problems related to finger correction machine and NOK_ML in case of faulty during marking or laser phases, while they are moved to NOK_C with gripper 1 if functional control machine detect some errors.

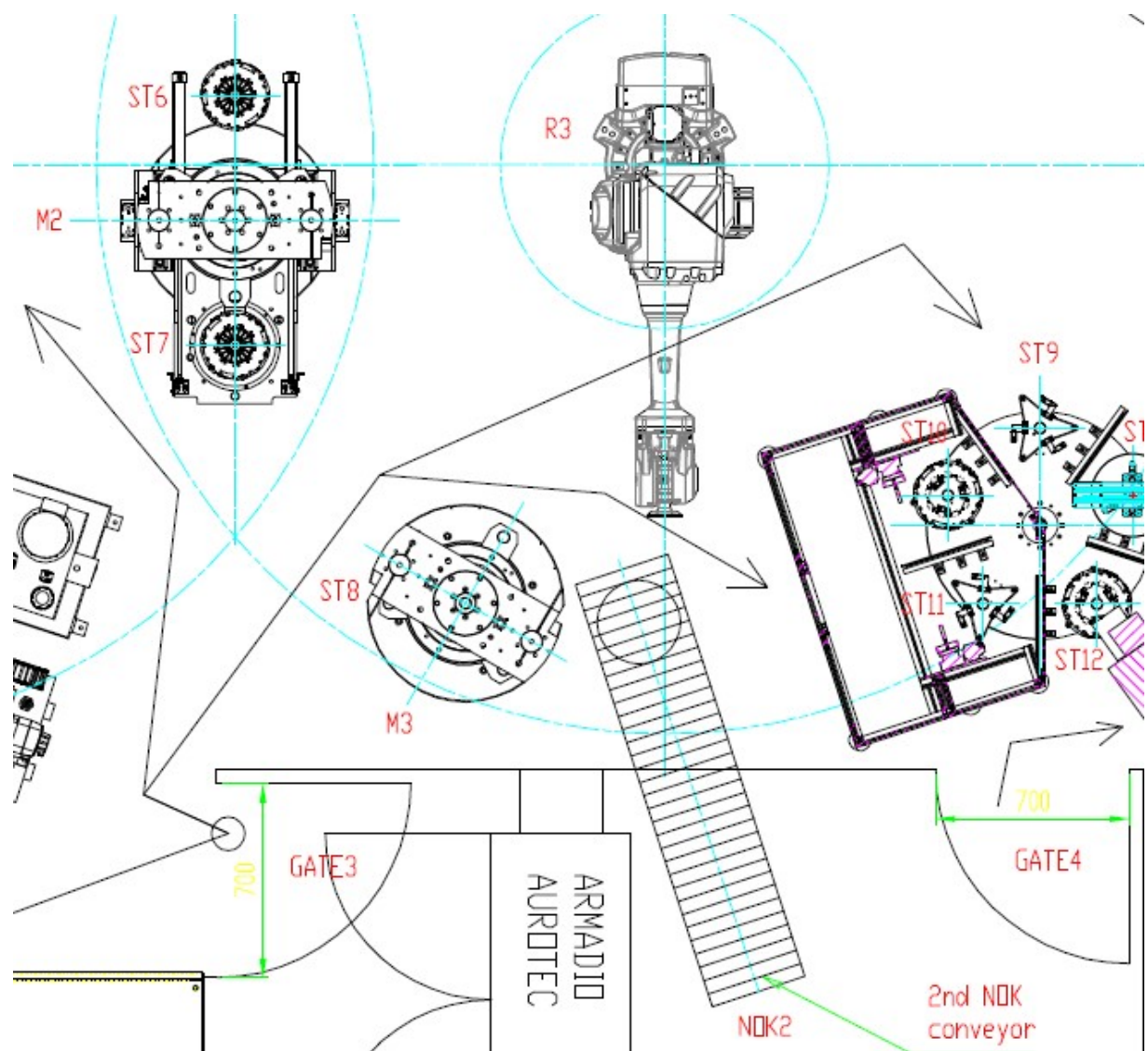


Figure 22: R3 task description

Also this time for programming clarity it was necessary to define the grippers as two different tools, so that also the pick and place procedures could be written in a specific way for each of them.

```
TASK PERS tooldata tGripper1=[TRUE, [[-355,-35,175],[1,0,0,0]], [31.5, [-6.4, -1.9, 64.7], [1,0,0,0], 0, 2.747, 3.586]];
TASK PERS tooldata tGripper2=[TRUE, [[355,35,175],[1,0,0,0]], [31.5, [-6.4, -1.9, 64.7], [1,0,0,0], 0, 2.747, 3.586]];
```

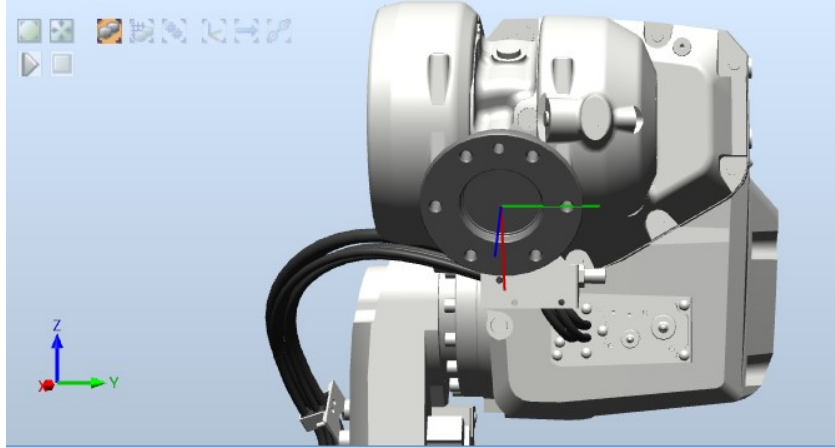


Figure 23: TCP reference frame with respect to the base one

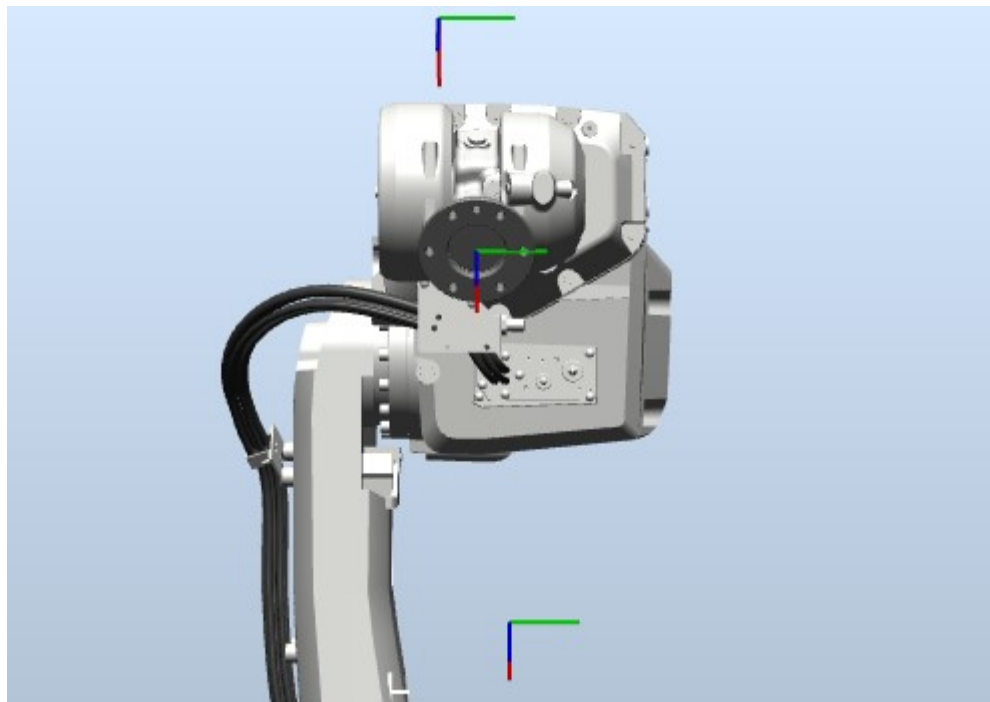


Figure 24: gripper 1 and 2 reference frames with respect to the TCP

2.3.1. I/O CONFIGURATION

As I did before, also for the I/O system of R3 robot I added the Profinet option inside the configuration section of RobotStudio, reserving 16 bytes of Digital Inputs and 16 bytes of Digital Outputs, as I did also for R1. All those signals are exchanged during the cycle with the PLC, particularly all the inputs are commands from the PLC to the robot related to the system state or to the movement it can do, while the outputs represent the confirmation that the commands have been received or that the movements have been executed.

For what concerns the system signal, there are different types of stop to be able to differentiate situations. Particularly the *stop_program* is the flex pendant signal, from which brakes are immediately activated, while *quick_stop* and *soft_stop* interrupt the program execution braking the motors with different deceleration ramps. Moreover, the difference between the two is related to the fact that when *quick_stop* is activated, all the motors are deactivated for security reasons, viceversa this does not happen when *soft_stop* signal becomes high. *Stop_end_of_cycle*, instead, can be activated just if both the grippers are empty, while *stop_end_of_instruction* is just related to the end of a line of code or movement.

Since many types of clutches can be tested inside the line, it was necessary to take into account the fact that each one has its own dimensions and height. This particularly will interfere with the place position of the rivets, which can change both along the radial position and the vertical one, but can be important in case also the position of the mechanical plugs changes. To be able to manage directly all these mechanical information, the variable *reference* was created as a number which can assume all the values between 0 and 127, each corresponding to an alphanumeric code translated by the PLC. This way, once the reference is defined and passed to the robot, it is able to perform the tasks without the need of other information. The only negative aspect is related to the fact that the WorkObjects must be defined in practice for each reference and this can take quite a long time, depending on the number of possible references.

Moreover, the same reference bits that are received as inputs are also set as output during the program execution, so the correctness of the signals exchanged can be easily checked.

Configurazione

Visualizza 1

Editor segnale X

Cerca

Cerca

4600-106813 (Stazione)

Comunicazione

I/O System

DeviceNet

PROFINET

Device

PN_Internal_Device

Local

EtherNetIP

SC_Feedback_Net

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Access Level	Default
Drive_on	PN_Internal_Device	Digital Input	0			Default	0
Start	PN_Internal_Device	Digital Input	1			Default	0
Stop_program	PN_Internal_Device	Digital Input	2			Default	0
Soft_stop	PN_Internal_Device	Digital Input	3			Default	0
Quick_stop	PN_Internal_Device	Digital Input	4			Default	0
Start_main	PN_Internal_Device	Digital Input	5			Default	0
Reset_emergency_stop	PN_Internal_Device	Digital Input	6			Default	0
Reset_error_signal	PN_Internal_Device	Digital Input	7			Default	0
Stop_end_of_cycle	PN_Internal_Device	Digital Input	8			Default	0
Stop_end_of_instruction	PN_Internal_Device	Digital Input	9			Default	0
System_restart	PN_Internal_Device	Digital Input	10			Default	0
PP_main	PN_Internal_Device	Digital Input	11			Default	0
Emergency_off	PN_Internal_Device	Digital Input	24			Default	0
Auto_mode	PN_Internal_Device	Digital Input	25			Default	0
Home_position_request	PN_Internal_Device	Digital Input	26			Default	0
Pick_S06_G1	PN_Internal_Device	Digital Input	32			Default	0
Pick_S07_G1	PN_Internal_Device	Digital Input	33			Default	0
Pick_S08_G2	PN_Internal_Device	Digital Input	34			Default	0
Place_S08_G1	PN_Internal_Device	Digital Input	35			Default	0
Place_S09_G2	PN_Internal_Device	Digital Input	36			Default	0
Pick_NOK_S09_G2	PN_Internal_Device	Digital Input	37			Default	0
Place_NOK_FC_G1	PN_Internal_Device	Digital Input	38			Default	0
Place_NOK_C_G2	PN_Internal_Device	Digital Input	39			Default	0
Place_NOK_ML_G1	PN_Internal_Device	Digital Input	40			Default	0
Open_gripper1	PN_Internal_Device	Digital Input	48			Default	0
Clutch_in_gripper1	PN_Internal_Device	Digital Input	49			Default	0
Open_gripper2	PN_Internal_Device	Digital Input	50			Default	0
Clutch_in_gripper2	PN_Internal_Device	Digital Input	51			Default	0
Bit0_reference	PN_Internal_Device	Digital Input	56			Default	0
Bit1_reference	PN_Internal_Device	Digital Input	57			Default	0
Bit2_reference	PN_Internal_Device	Digital Input	58			Default	0
Bit3_reference	PN_Internal_Device	Digital Input	59			Default	0
Bit4_reference	PN_Internal_Device	Digital Input	60			Default	0
Bit5_reference	PN_Internal_Device	Digital Input	61			Default	0
Bit6_reference	PN_Internal_Device	Digital Input	62			Default	0
Bit7_reference	PN_Internal_Device	Digital Input	63			Default	0

Figure 25: input configuration

Configurazione

Visualizza1

Editor segnale x

Cerca

Cerca

4600-106813 (Stazione)

Comunicazione

I/O System

DeviceNet

PROFINET

Device

PN_Internal_Device

Local

EtherNetIP

SC_Feedback_Net

Name	Assigned to Device	Type of Signal	Device Mapping	Signal Identification Label	Category	Access Level	Default
Drive_on_ok	PN_Internal_Device	Digital Output	0			Default	0
Cycle_running	PN_Internal_Device	Digital Output	1			Default	0
Emergency_stop	PN_Internal_Device	Digital Output	2			Default	0
Error_signal	PN_Internal_Device	Digital Output	3			Default	0
Robot_in_home_pos	PN_Internal_Device	Digital Output	26			Default	0
Clutch_picked_from_S06	PN_Internal_Device	Digital Output	32			Default	0
Clutch_picked_from_S07	PN_Internal_Device	Digital Output	33			Default	0
Clutch_picked_from_S08	PN_Internal_Device	Digital Output	34			Default	0
Clutch_placed_in_S08	PN_Internal_Device	Digital Output	35			Default	0
Clutch_placed_in_S09	PN_Internal_Device	Digital Output	36			Default	0
Clutch_NOK_picked_from_S09	PN_Internal_Device	Digital Output	37			Default	0
Clutch_placed_in_NOK_conveyor	PN_Internal_Device	Digital Output	38			Default	0
Out_of_area_S06	PN_Internal_Device	Digital Output	40			Default	0
Out_of_area_S07	PN_Internal_Device	Digital Output	41			Default	0
Out_of_area_S08	PN_Internal_Device	Digital Output	42			Default	0
Out_of_area_S09	PN_Internal_Device	Digital Output	43			Default	0
Out_of_area_NOK_conveyor	PN_Internal_Device	Digital Output	44			Default	0
Gripper1_opened	PN_Internal_Device	Digital Output	48			Default	0
Gripper1_closed	PN_Internal_Device	Digital Output	49			Default	0
Gripper2_opened	PN_Internal_Device	Digital Output	50			Default	0
Gripper2_closed	PN_Internal_Device	Digital Output	51			Default	0
Bit0_clutch_reference	PN_Internal_Device	Digital Output	56			Default	0
Bit1_clutch_reference	PN_Internal_Device	Digital Output	57			Default	0
Bit2_clutch_reference	PN_Internal_Device	Digital Output	58			Default	0
Bit3_clutch_reference	PN_Internal_Device	Digital Output	59			Default	0
Bit4_clutch_reference	PN_Internal_Device	Digital Output	60			Default	0
Bit5_clutch_reference	PN_Internal_Device	Digital Output	61			Default	0
Bit6_clutch_reference	PN_Internal_Device	Digital Output	62			Default	0
Bit7_clutch_reference	PN_Internal_Device	Digital Output	63			Default	0

Figure 26: output configuration

3. PICK AND PLACE TASK

Since the grippers have been considered as two different tools, for each situation it was possible to create ad hoc pick and place functions.

Then, for all the robots a home and wait position have been defined. In home position the robot is moved so that the first three links are in vertical position, while the last two are in horizontal one: this allows the operator to correctly calibrate the tool attached to the TCP. However, due to the fact that the line is quite crowded of machines which don't allow the robot to move freely, the home position for the calibration has been divided in two subfunction. The first guarantees the correct position of 0° for that joints that are able to reach it without crashing. Then, a second function is used to move to the zero position the others, keeping the calibrated one in whatever combination so that the robot is out of encumbrance while the last joints are calibrated. The wait position, instead, has been created in case the task of the robot is quicker with respect to the flow rate of the clutches along the line. Moving the robot in wait position, it is positioned near the area where next task have to be performed, usually the pick task of a new item which enters its work area, so without interfering with the machines, but reducing cycle time.

All instructions have been performed approaching the item from the vertical direction. This means that the pick positions have been stored as the points that lies on the vertical with respect to the pick point of the item, but their distance can change accordingly to the situation. So the complete movement is divided in 2 steps:

- First of all the robot is moved so that the center of the interested gripper is positioned at a certain height from the item it has to pick up. Distance is related particularly to the double gripper dimensions and encumbrance, together with the characteristics of the pick place (for example, to perform the pick task inside the machines, it is necessary to take into account also the presence of the drawer and of its mechanical guides).

```
MoveL Offs(pick_gripper1_TOX,0,0,100), v1000, z50, tGripper1;
```

This movement has been performed trough the *MoveL* command adding an offset of 100 mm inside the third field of *Offs* instruction, which is the value related to z axis. Moreover, the parameter *z50* has been added. It represents the radius length of a theoric circle centered in the target point. This means that at maximum the TCP at the end of the movement can lie on an area which is far at maximum 50 mm from the target point.

- Then gripper is finally approached to the item:

```
MoveL pick_gripper1_TOX,v100,fine,tGripper1;
```

This time, since no errors are allowed when picking the item, it is necessary to add the parameter *fine* to be able to guarantee as much precision as possible.

- Finally, once the item is safely fixed inside the gripper and can be moved, it is necessary to perform another *MoveL* instruction along the vertical direction as before:

```
MoveL Offs(pick_gripper1_TOX,0,0,100), v1000, z50, tGripper1;
```

Also for place tasks the approaching movement is a vertical one, due to the fact that it is necessary to avoid collisions with other objects and because often there is not so much space, so not every type of movement is allowed. Moreover the gripper has to be parallel to the pick and place positions to be sure to pick at the same time and with the same force the clutch mechanism using three mechanical plugs positioned at 120°. For what concerns robot R2, again it has to approach the hole for rivet parallel to the mechanism due to the cover of the mechanism itself which is higher with respect to the hole surface and not regular. So it is better to tilt the arm, having just the wrist oriented so that the gripper is in vertical position. This way it is possible to gain also cycle time, because the robot can be more distant and does not need to move near the place area. This movement is composed by 3 instruction as follows:

- The first instruction is used to move the robot at a vertical distance equal to 100 mm from the place area, as before. The only difference consist on the fact that the robot arm now is no more in vertical position, but tilted:

```
MoveL Offs(place_gripper1_S05,0,0,100),v1000,z50,tGripper1;
```

To guarantee good precision, z50 option has been added.

- Then the arm is approached along the vertical direction decreasing the speed and the horizontal adjustment to increase precision:

```
MoveL offs(place_gripper1_S05,0,0,0),v100,fine,tGripper1;
```

- To go away after the item has been placed, I added an offset to the removal position, then the gripper can move away and it is ready to perform the next task.

```
MoveL Offs(place_gripper1_S05,0,0,100),v1000,z50,tGripper1;
```

Now it is possible to analyze in detail the procedures of each robot.

3.1. ROBOT R1

For robot R1 the pick and place tasks have been defined inside the module called *pickPlace*. This module contains many procedures, one for each pick and one for each place task that has to be performed by gripper 1 or by gripper 2.

Each of them has been defined in the same way: this means that the only differences are related to the pick and place positions, while the structure is almost the same. Moreover, due to the fact that in practice there are many obstacles, it was necessary to add some fly points as points where the robot has to pass in order both to avoid collisions or singularities. Particularly, the second ones are the most critical issues: they appear either when axis 4, 5 and 6 are alligned or when axis 5 is alligned with the base. If the robot moves accidentally in one of these situations, then it is not able to move anymore, due to the fact that these configurations, from the theoretical point of view, represent position where joint velocities become infinite.

3.1.1. PICK_GRIPPER1_ENTRY_LINE

The procedure is called every time there is the need to pick a clutch from the entry line and the operation is performed by gripper 1, as the name suggests. Once it is called, the outputs *Out_of_area_entry_line* and *Robot_in_home_pos* are set to 0 when the robot enters a pre-defined zone, to indicate that the robot could interfere with the entry line area, so it is not out of its area, and that it is no more in home position since it will start to move soon. Particularly, these signals are managed by the so called world zones, as I will explain in the next chapter. Then a linear movement to *pick_gripper1_TOX* is performed until gripper 1 is 100 mm above the position where the clutch will be picked up. To check the possibility to pick a new item, it is

necessary to check both the gripper status and that a no clutches are inside it. If everything is ok, a *fine* linear movement is performed to approach the clutch, then gripper 1 is closed. At this point the program stops for a while, waiting for the input signal *Clutch_in_gripper1* to become active, indicating that the pick task has been performed successfully. If this does not happen, then an error message is generated on the Flex Pendant, while the boolean variable *closingGripper1Error* becomes true; otherwise the arm is moved along the vertical position and next task can be executed.

```

PROC pick_gripper1_entry_line()

IF (place_to_ST6_G2 = TRUE OR place_to_ST7_G2 = TRUE) THEN
MoveL [[225.03,486.98,1262.64],[0.00538369,-0.00539757,0.999971,0.000565707],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
MoveL [[478.24,-38.21,1237.99],[0.00523205,-0.798493,-0.601979,0.00187871],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
ENDIF
MoveL Offs(pick_gripper1_TOX,0,0,100), v1000, z50, tGripper1;                                !move the robot at 100mm above from the netry line
IF DOutput(Gripper1_closed) = 0 AND DOutput(Gripper1_opened) = 1 AND Clutch_in_gripper1 = 0 THEN !check the absence of clutch in gripper1
    MoveL pick_gripper1_TOX,v100,fine,tGripper1;
    SetDO Gripper1_closed, 1;                                                                !gripper 1 is closed
    SetDO Gripper1_opened, 0;
    WaitUntil DInput(Clutch_in_gripper1) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
ENDIF
IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper1_TOX,0,0,100), v100, z50, tGripper1;                            !move the robot at 100mm above from the netry line
    SetDO Error_signal,1;
    TPErase;                                                                                !erase on the Flex pendant the previous message
    TPWrite closing_gripper1_error;                                                         !show the error
    closingGripper1Error := TRUE;                                                            !set the error flag
    RETURN;
ELSE
    SetDO Clutch_picked_from_entry_line, 1;
    MoveL Offs(pick_gripper1_TOX,0,0,100),v100,z50,tGripper1;
    closingGripper1Error := FALSE;
    place_to_ST6_G2 := FALSE;
    place_to_ST7_G2 := FALSE;
    place_to_NOK_B_conv_G2 := FALSE;
    pick_from_STA_G1 := TRUE;
ENDIF
ENDPROC

```

Figure 27: pick task from entry line performed by gripper 1

3.1.2. PICK_GRIPPER2_ST5

```

PROC pick_gripper2_ST5()

MoveL Offs(pick_gripper2_S05,0,0,50), v1000, z50, tGripper2;
IF DOutput(Gripper2_closed) = 0 AND DOutput(Gripper2_opened) = 1 AND Clutch_in_gripper2 = 0 THEN
    MoveL pick_gripper2_S05,v100,fine,tGripper2;
    SetDO Gripper2_closed, 1;
    SetDO Gripper2_opened, 0;
    WaitUntil DInput(Clutch_in_gripper2) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
ENDIF
IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper2_S05,0,0,50), v100, z50, tGripper2;
    SetDO Error_signal,1;
    TPErase;
    TPWrite closing_gripper2_error;
    closingGripper2Error := TRUE;
    RETURN;
ELSE
    SetDO Clutch_picked_from_S05, 1;
    MoveL Offs(pick_gripper2_S05,0,0,50),v100,z50,tGripper2;
    closingGripper2Error := FALSE;
    pick_from_ST5_G2 := TRUE;
ENDIF
ENDPROC

```

Figure 28: pick task from ST5 performed by gripper 2

3.1.3. PLACE_GRIPPER1_ST5

With procedure *place_gripper1_ST5()*, robot R1 moves towards ST5. Once the function is called *Robot_in_home_pos* and *Out_of_ST5* are set to 0 to signal the movement, then robot starts to approach ST5 position. As described before, the motion is composed by 3 steps to guarantee as much precision as possible. Once the robot has reached ST5 position, if the command from PLC *Open_gripper1* is set to 1, then the gripper is opened. To be sure that the operation has been performed correctly, the program waits for a while. Finally the opening task is checked: if no errors are detected the signal *Clutch_placed_in_S05* is set to 1 and the robot starts moving away, out of area of ST5; otherwise an error is displayed on the Flex Pendant once the *OpeningGripper1Error* signal is set to 1.

```
PROC place_gripper1_ST5()

  MoveL Offs(place_gripper1_S05,0,0,100),v1000,z50,tGripper1;
  MoveL offs(place_gripper1_S05,0,0,0),v100,fine,tGripper1;
  IF Open_gripper1 = 1 THEN
    SetDO Gripper1_opened, 1;
    SetDO Gripper1_closed, 0;
  ENDIF
  WaitUntil DInput(Clutch_in_gripper1) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper1_error;
    openingGripper1Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_S05, 1;
    MoveL offs(place_gripper1_S05,0,0,100),v100,z50,tGripper1;
    openingGripper1Error := FALSE;
    pick_from_STA_G1 := FALSE;
    place_to_ST5_G1 := TRUE;
  ENDIF
ENDPROC
```

Figure 29: place task to ST5 performed by gripper 1

3.1.4. PLACE_GRIPPER1_NOK1_A_CONVEYOR

```
PROC place_gripper1_NOK1_A_conveyor()

  MoveL Offs(place_gripper1_NOK1_A,0,0,100),v1000,z50,tGripper1;
  MoveL offs(place_gripper1_NOK1_A,0,0,0),v100,fine,tGripper1;
  IF Open_gripper1 = 1 THEN
    SetDO Gripper1_opened, 1;
    SetDO Gripper1_closed, 0;
  ENDIF
  WaitUntil DInput(Clutch_in_gripper1) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper1_error;
    openingGripper1Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_NOK1_A, 1;
    MoveL Offs(place_gripper1_NOK1_A,0,0,100),v100,z50,tGripper1;
    openingGripper1Error := FALSE;
    pick_from_STA_G1 := FALSE;
    place_to_NOK_A_conv_G1 := TRUE;
  ENDIF
ENDPROC
```

Figure 30: place task to NOK1_A conveyor performed by gripper 1

3.1.5. PLACE_GRIPPER2_ST6

```

PROC place_gripper2_ST6()

IF pick_from_ST5_G2 = TRUE THEN
  MoveL [[225.03,486.98,1262.64],[0.00538369,-0.00539757,0.999971,0.000565707],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
  MoveL [[-618.41,553.84,1253.69],[0.000820031,-0.999147,0.0409407,-0.00536257],[1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
ENDIF
MoveL Offs(place_gripper2_S06,0,0,110),v1000,z50,tGripper2;
MoveL offs(place_gripper2_S06,0,0,0),v100,fine,tGripper2;
IF Open_gripper2 = 1 THEN
  SetDO Gripper2_opened, 1;
  SetDO Gripper2_closed, 0;
ENDIF
WaitUntil DInput(Clutch_in_gripper2) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
IF bTimeOut THEN
  !Error
  SetDO Error_signal, 1;
  TPErase;
  TPWrite opening_gripper2_error;
  openingGripper2Error := TRUE;
  RETURN;
ELSE
  SetDO Clutch_placed_in_S06, 1;
  MoveL Offs(place_gripper2_S06,0,0,100),v100,z50,tGripper2;
  MoveL [[-618.41,553.84,1253.69],[0.000820031,-0.999147,0.0409407,-0.00536257],[1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
  openingGripper2Error := FALSE;
  pick_from_ST5_G2 := FALSE;
  place_to_ST6_G2 := TRUE;
ENDIF
ENDPROC

```

!erase on the Flex pendant the previous message
!show the error
!set the error flag

!clutch placed in S06
!go away from clutch

Figure 31: place task to ST6 performed by gripper 2

3.1.6. PLACE_GRIPPER2_ST7

```

PROC place_gripper2_ST7()

MoveL Offs(place_gripper2_S07,0,0,100),v1000,z50,tGripper2;
MoveL offs(place_gripper2_S07,0,0,0),v100,fine,tGripper2;
IF Open_gripper2 = 1 THEN
  SetDO Gripper2_opened, 1;
  SetDO Gripper2_closed, 0;
ENDIF
WaitUntil DInput(Clutch_in_gripper2) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
IF bTimeOut THEN
  !Error
  SetDO Error_signal, 1;
  TPErase;
  TPWrite opening_gripper2_error;
  openingGripper2Error := TRUE;
  RETURN;
ELSE
  SetDO Clutch_placed_in_S07, 1;
  MoveL Offs(place_gripper2_S07,0,0,100),v1000,fine,tGripper2;
  MoveL Offs(place_gripper2_S07,0,-300,100), v100, fine, tGripper2;
  openingGripper2Error := FALSE;
  pick_from_ST5_G2 := FALSE;
  place_to_ST7_G2 := TRUE;
ENDIF
ENDPROC

```

Figure 32: place task to ST7 performed by gripper 2

3.1.7. PLACE_GRIPPER2_NOK1_B_CONVEYOR

```

PROC place_gripper2_NOK1_B_conveyor()

    MoveL Offs(place_gripper2_NOK1_B,0,0,100),v1000,z50,tGripper2;
    MoveL offs(place_gripper2_NOK1_B,0,0,0),v100,fine,tGripper2;
    IF Open_gripper2 = 1 THEN
        SetDO Gripper2_opened, 1;
        SetDO Gripper2_closed, 0;
    ENDIF
    WaitUntil DInput(Clutch_in_gripper2) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
    IF bTimeOut THEN
        !Error
        SetDO Error_signal,1;
        TPErase;
        TPWrite opening_gripper2_error;
        openingGripper2Error := TRUE;
        RETURN;
    ELSE
        SetDO Clutch_placed_in_NOK1_B, 1;
        MoveL Offs(place_gripper2_NOK1_B,0,0,100),v100,z50,tGripper2;
        openingGripper2Error := FALSE;
        pick_from_ST5_G2 := FALSE;
        place_to_NOK_B_conv_G2 := TRUE;
    ENDIF
ENDPROC

```

Figure 33: place task to NOK1_B conveyor performed by gripper 2

3.2. ROBOT R2

For robot R2 the pick and place tasks have been defined inside the module called *pickPlace*. This module contains in total 4 procedures, one for pick and one for place related to gripper 1 and other two related to gripper 2.

3.2.1. PICK1

Pick1() procedure, as the name suggests, describes the pick task performed by gripper 1. Once it is called, the outputs *Out_of_rivet_feeder* and *Robot_in_home_pos* are set to 0, to indicate that the robot could interfere with the rivet feeder area, so it is not out of its area, and that it is no more in home position since it will start to move soon. Then a linear movement to *feederPos_gripper1* is performed until the robot is 50 mm above the position where the rivet will be picked up.

Moreover, since the rivet feeder is composed by 14 linear guides which move rivets of different weight and dimensions, there are 14 possible positions where it can be asked to pick up a rivet. To manage this information, the variable *rivet1code* has been introduced as the decimal number which comes from the conversion of the binary variables from *Bit0_rivet1* to *Bit3_rivet1* as follows:

```
!Compute_pos_rivet1;                                !compute where to pick the rivet
rivet1code := 8*DInput(Bit3_rivet1)+4*DInput(Bit2_rivet1)+2*DInput(Bit1_rivet1)+Dinput(Bit0_rivet1);
IF rivet1code < 14 THEN
    rivet1Error := FALSE;
    pick1;                                           !make the vacuum to take a rivet
    rivet_code_gripper1;                           !update the code of the rivet on gripper 1
    rivet1_code_control;                           !check the correctness of rivet code
ELSE
    TPErase;
    TPWrite rivet1_uncorrect;
    rivet1Error := TRUE;
ENDIF
```

Figure 34: position of rivet 1 computation

Once the variable is computed, its correctness is checked (since there are at maximum 14 positions it could not be greater than 14). If everything is ok, then *pick1* procedure is called; otherwise an error on the Flex Pendant is generated.

When the robot reach the desired position above the rivet feeder, if the gripper is empty, it can approach the rivet with a *fine* movement. Then the program will activate the vacuum and wait for a while. Finally vacuum activation is checked: if the generation has been correctly performed, then the output *Rivet1_picked* is set to 1 to indicate that the rivet has been picked up and the robot starts moving away from rivet feeder area. If something wrong happens, instead, an error is generated and a message is displayed on the Flex Pendant.

```

PROC pick1()
  MoveL Offs(feederPos_gripper1{rivet1code},0,0,50), v500, z20, tGripper1;
  IF DOutput(Vacuum_activation_on_gripper1) = 0 AND Rivet1_in_gripper1 = 0 THEN
    MoveL feederPos_gripper1{rivet1code},v1000,fine,tGripper1;
    SetDO Vacuum_activation_on_gripper1, 1;
    WaitUntil DInput(Rivet1_in_gripper1) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
    IF bTimeOut THEN
      !Error
      MoveL Offs(feederPos_gripper1{rivet1code},0,0,50),v1000,z10,tGripper1;
      SetDO Error_signal,1;
      TPErase;
      TPWrite vacuum_error1;
      bVacuum1Error := TRUE;
      RETURN;
    ELSE
      SetDO Rivet1_picked, 1;
      MoveL Offs(feederPos_gripper1{rivet1code},0,0,50),v1000,z10,tGripper1;
      bVacuum1Error := FALSE;
    ENDIF
  ENDIF
ENDPROC

```

Figure 35: pick1 procedure performed by robot R2

3.2.2. PICK2

The same considerations can be done also for what concerns gripper 2, as follows:

```

!Compute_pos_rivet2;                                !compute where to pick the rivet
rivet2code := 8*DInput(Bit3_rivet2)+4*DInput(Bit2_rivet2)+2*DInput(Bit1_rivet2)+Dinput(Bit0_rivet2);
IF rivet2code < 14 THEN
  rivet2Error := FALSE;
  pick2;                                              !make the vacuum to take a rivet
  rivet_code_gripper2;                               !update the code of the rivet on gripper 2
  rivet2_code_control;                              !check the correctness of rivet code
ELSE
  TPErase;
  TPWrite rivet2_uncorrect;
  rivet2Error := TRUE;
ENDIF

```

Figure 36: position of rivet 2 computation

```

PROC pick2()

MoveL Offs(feederPos_gripper2{rivet2code},0,0,50), v500, z20, tGripper2;
IF DOutput(Vacuum_activation_on_gripper2) = 0 AND Rivet2_in_gripper2 = 0 THEN
    MoveL feederPos_gripper2{rivet2code},v1000,fine,tGripper2;
    SetDO Vacuum_activation_on_gripper2, 1;
    WaitUntil DInput(Rivet2_in_gripper2) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
    IF bTimeOut THEN
        !Error
        MoveL Offs(feederPos_gripper2{rivet2code},0,0,50),v1000,z10,tGripper2;
        SetDO Error_signal,1;
        TPErase;
        TPWrite vacuum_error2;
        bVacuum2Error := TRUE;
        RETURN;
    ELSE
        SetDO Rivet2_picked, 1;
        MoveL Offs(feederPos_gripper2{rivet2code},0,0,50),v1000,z10,tGripper2;
        bVacuum2Error := FALSE;
    ENDIF
ENDIF
ENDPROC

```

Figure 37: pick2 procedure performed by robot R2

3.2.3. PLACE1

With procedure *place1()*, robot R2 moves toward the cover of the clutch present in ST2. Once the function is called *Robot_in_home_pos* and *Out_of_ST2* are set to 0 to signal the movement, then robot starts to approach ST2 position. As described before, the motion is composed by 3 steps, all referred to the same type of clutch through the variable *reference*. Its computation is performed inside the system module *reference_choice()* in the procedure *ref()*, which is called inside the *MainModule()* at restart. This way, since there is a position for each type of clutch, all the dimensions changes are automatically taken into account without the need of select them manually.

Once the robot has reached ST2 position, the deactivation of vacuum is set forcing to 0 the signal *Vacuum_activation_on_gripper2*, then the program will wait for a while to be sure that the rivet is no more attached to the gripper. Finally the deactivation is checked: if no errors are detected the signal *Rivet1_placed* is set to 1 and the robot starts moving away, out of area of ST2; otherwise an error is displayed on the Flex Pendant once the *Error_signal* is set to 1.

```

PROC place1()
  MoveJ Offs(place_pos_g1{rivet1code},0,0,50), v5000, z20, tGripper1;
  MoveL place_pos_g1{rivet1code}, v100, fine, tGripper1;
  WaitTime 0.5;
  SetDO Vacuum_activation_on_gripper1, 0;
  WaitUntil DOutput(Vacuum_activation_on_gripper1) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal,1;
    TPErase;
    TPWrite place_error1;
    bVacuum1Error := TRUE;
    MoveL Offs(place_pos_g1{rivet1code},0,0,50), v500, z20, tGripper1;
    RETURN;
  ELSE
    SetDO Rivet1_placed, 1;
    MoveL Offs(place_pos_g1{rivet1code},0,0,50), v500, z20, tGripper1;
    bVacuum1Error := FALSE;
  ENDIF
ENDPROC

```

Figure 38: place 1 procedure performed by gripper 1

3.2.4. PLACE2

Place2() procedure performs exactly the same operations performed by *place1()*, the only difference consists on the fact that now tool gripper 2 is used.

```

PROC place2()
  MoveJ Offs(place_pos_g2{rivet2code},0,0,50), v5000, z20, tGripper2;
  MoveL place_pos_g2{rivet2code}, v100, fine, tGripper2;
  WaitTime 0.5;
  SetDO Vacuum_activation_on_gripper2, 0;
  WaitUntil DOutput(Vacuum_activation_on_gripper2) = 0\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal,1;
    TPErase;
    TPWrite place_error2;
    bVacuum2Error := TRUE;
    MoveL Offs(place_pos_g2{rivet2code},0,0,50), v500, z20, tGripper2;
    RETURN;
  ELSE
    SetDO Rivet2_placed, 1;
    bVacuum2Error := FALSE;
    MoveL Offs(place_pos_g2{rivet2code},0,0,50), v500, z20, tGripper2;
  ENDIF
ENDPROC

```

Figure 39: place2 procedure performed by robot R2

3.3. ROBOT R3

Also for robot R3 the pick and place tasks have been defined inside the module called *pickPlace*. This module contains many procedures, one for each pick and one for each place task that has to be performed by gripper 1 or by gripper 2.

Each of them has been defined in the same way: this means that the only differences are related to the pick and place positions, while the structure is almost the same. Moreover, due to the fact that in practice there are many obstacles, it was necessary to add some fly points as points where the robot has to pass in order both to avoid collisions or singularities. Particularly, the second ones are the most critical issues: they appear either when axis 4, 5 and 6 are aligned or when axis 5 is aligned with the base. If the robot moves accidentally in one of these situations, then it is not able to move anymore, due to the fact that these configurations, from the theoretical point of view, represent position where joint velocities become infinite.

3.3.1. PICK_GRIPPER2_ST6

The procedure is called every time there is the need to pick a clutch from ST6 and the operation is performed by gripper 2, as the name suggests. Once it is called, the outputs *Out_of_area_ST6* and *Robot_in_home_pos* are set to 0, to indicate that the robot could interfere with ST6 area, so it is not out of its area, and that it is no more in home position since it will start to move soon. Then a linear movement to *pick_gripper2_ST6* is performed until gripper 1 is 100 mm above the position where the clutch will be picked up. To check the possibility to pick a new item, it is necessary to control both the gripper status and that no clutches are inside it. If everything is ok, a *fine* linear movement is performed to approach the clutch, then gripper 2 is closed. At this point the program stops for a while, waiting for the input signal *Clutch_in_gripper1* to become active, indicating that the pick task has been performed successfully. If this does not happen, then an error message is generated on the Flex Pendant, while the boolean variable *closingGripper1Error* becomes true; otherwise the arm is moved along the vertical position and next task can be executed.

```
PROC pick_gripper2_ST6()
  MoveL [[655.84,-291.38,1206.98],[0.00151811,0.937365,-0.348345,0.000577677]],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09],v1000,z50,tool0;
  MoveL [[-305.90,-713.10,1105.31],[0.0034614,0.999021,-0.0433244,0.00828347]],[-1,-1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09],v1000,z50,tGripper2;
  MoveL Offs(pick_gripper2_S06,0,0,105.31), v1000, z50, tGripper2;
  IF DOutput(Gripper2_closed) = 0 AND DOutput(Gripper2_opened) = 1 AND Clutch_in_gripper2 = 0 THEN
    MoveL pick_gripper2_S06,v100,fine,tGripper2;
    SetDO Gripper2_closed, 1;
    SetDO Gripper2_opened, 0;
    WaitUntil DInput(Clutch_in_gripper2) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
  ELSE
    SetDO Error_signal,1;
  ENDIF
  IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper2_S06,0,0,105.31), v500, z20, tGripper2;
    SetDO Error_signal,1;
    TPErase;
    TPWrite closing_gripper2_error;
    closingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_picked_from_S06, 1;
    MoveL Offs(pick_gripper2_S06,0,0,105.31),v100,z50,tGripper2;
    MoveL [[-659.32,-646.18,1274.03],[0.00825686,0.0299461,0.999511,-0.0035599]],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09], v1000, z50, tool0;
    closingGripper2Error := FALSE;
    pick_from_ST6_G2 := TRUE;
    place_to_ST9_G1 := FALSE;
    place_to_NOK_M_G2 := FALSE;
    place_to_NOK_C_G1 := FALSE;
  ENDIF
```

Figure 40: pick task from ST6 performed by gripper 2

3.3.2. PICK_GRIPPER2_ST7

```

PROC pick_gripper2_ST7()
  MoveL [[830.03,-994.81,1194.63],[0.00278644,0.843161,-0.537649,0.00241413],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper2;
  MoveL Offs(pick_gripper2_S07,0,0,158.7), v1000, z50, tGripper2;
  IF DOutput(Gripper2_closed) = 0 AND DOutput(Gripper2_opened) = 1 AND Clutch_in_gripper2 = 0 THEN
    MoveL pick_gripper2_S07,v100,fine,tGripper2;
    SetDO Gripper2_closed, 1;
    SetDO Gripper2_opened, 0;
    WaitUntil DInput(Clutch_in_gripper2) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
  ELSE
    SetDO Error_signal,1;
  ENDIF
  IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper2_S07,0,0,158.7), v100, z50, tGripper2;
    SetDO Error_signal,1;
    TPErase;
    TPWrite closing_gripper2_error;
    closingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_picked_from_S07, 1;
    MoveL Offs(pick_gripper2_S07,0,0,158.7),v100,z50,tGripper2;
    MoveL [[830.03,-994.81,1194.63],[0.00278644,0.843161,-0.537649,0.00241413],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper2;
    closingGripper2Error := FALSE;
    pick_from_ST7_G2 := TRUE;
    place_to_ST9_G1 := FALSE;
    place_to_NOK_M_G2 := FALSE;
    place_to_NOK_C_G1 := FALSE;
  ENDIF
ENDPROC

```

Figure 41: pick task from ST7 performed by gripper 2

3.3.3. PICK_GRIPPER1_ST8

```

PROC pick_gripper1_ST8()
  IF pick_from_ST6_G2 = TRUE THEN
    MoveL [[1040.65,-358.30,1032.27],[0.000537476,0.295671,0.955288,-0.00154856],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
    MoveL [[1040.66,-358.30,1032.28],[0.000541685,0.29567,0.955289,-0.00154799],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
    pick_gripper1_S08 := [[1525.96,-690.69,984.52],[0.000544693,0.295671,0.955289,-0.00154538],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  ELSEIF pick_from_ST7_G2 = TRUE THEN
    MoveL [[546.63,-613.08,1366.49],[0.0009732,0.129335,0.991594,-0.0035285],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
    MoveL [[1040.66,-358.30,1032.28],[0.000541685,0.29567,0.955289,-0.00154799],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
    pick_gripper1_S08 := [[1525.96,-690.69,984.52],[0.000544693,0.295671,0.955289,-0.00154538],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  ENDIF
  MoveL Offs(pick_gripper1_S08,0,0,15,46.94), v1000, z50, tGripper1;
  IF DOutput(Gripper1_closed) = 0 AND DOutput(Gripper1_opened) = 1 AND Clutch_in_gripper1 = 0 THEN
    MoveL pick_gripper1_S08,v100,fine,tGripper1;
    SetDO Gripper1_closed, 1;
    SetDO Gripper1_opened, 0;
    WaitUntil DInput(Clutch_in_gripper1) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
  ELSE
    SetDO Error_signal,1;
  ENDIF
  IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper1_S08,0,0,46.94), v500, z20, tGripper1;
    SetDO Error_signal,1;
    TPErase;
    TPWrite closing_gripper1_error;
    closingGripper1Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_picked_from_S08, 1;
    MoveL Offs(pick_gripper1_S08,0,0,46.94),v500,z50,tGripper1;
    IF pick_from_ST6_G2 = TRUE THEN
      MoveL [[1040.66,-358.30,1032.28],[0.000541685,0.29567,0.955289,-0.00154799],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
      MoveL [[655.84,-291.38,1206.98],[0.00151811,0.937365,-0.348345,0.000577677],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tool0;
    ELSEIF pick_from_ST7_G2 = TRUE THEN
      MoveL [[1040.66,-358.30,1032.28],[0.000541685,0.29567,0.955289,-0.00154799],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
    ENDIF
    closingGripper1Error := FALSE;
    pick_from_ST8_G1 := TRUE;
  ENDIF
ENDPROC

```

Figure 42: pick task from ST8 performed by gripper 1

3.3.3. PICK_GRIPPER2_NOK_ST9

```

PROC pick_gripper2_NOK_ST9()

  MoveL [[24.13,818.66,1185.25],[0.001203,0.992405,0.122978,0.00271267],[0,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
  MoveL [[259.68,1343.05,1083.99],[0.00125514,0.992403,0.122991,0.00271127],[1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
  MoveL Offs(pick_gripper2_NOK_S09,0,0,100), v1000, z50, tGripper2;
  IF DOutput(Gripper2_closed) = 0 AND DOutput(Gripper2_opened) = 1 AND Clutch_in_gripper2 = 0 THEN
    MoveL pick_gripper2_NOK_S09,v100,fine,tGripper2;
    SetDO Gripper2_closed, 1;
    SetDO Gripper2_opened, 0;
    WaitUntil DInput(Clutch_in_gripper2) = 1\MaxTime:=1\TimeFlag:=bTimeOut;
  ELSE
    SetDO Error_signal,1;
  ENDIF
  IF bTimeOut THEN
    !Error
    MoveL Offs(pick_gripper2_NOK_S09,0,0,100), v100, z50, tGripper2;
    SetDO Error_signal,1;
    TPErase;
    TPWrite closing_gripper2_error;
    closingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_picked_from_S08, 1;
    MoveL Offs(pick_gripper2_NOK_S09,50,0,200),v1000,z10,tGripper2;
    MoveL [[259.68,1343.05,1083.99],[0.00125514,0.992403,0.122991,0.00271127],[1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
    MoveL [[-46.49,1207.16,1186.27],[1.19926E-06,0.105797,-0.994388,2.91914E-05],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
    IF Place_S09_G1 = 0 THEN
      MoveL [[299.85,590.84,1012.24],[2.82203E-05,-0.956296,0.292387,-0.00295612],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
    ENDIF
    closingGripper2Error := FALSE;
    pick_from_ST9_G2 := TRUE;
  ENDIF
ENDPROC

```

Figure 43: pick task from ST9 performed by gripper 2

3.3.4. PLACE_GRIPPER2_ST8

With procedure *place_gripper1_ST8()*, robot R3 moves towards ST8. Once the function is called *Robot_in_home_pos* and *Out_of_ST8* are set to 0 to signal the movement, then robot starts to approach ST8 position. As described before, the motion is composed by 3 steps to guarantee as much precision as possible. Once the robot has reached ST8 position, if the command from PLC *Open_gripper2* is set to 1, then the gripper is opened. To be sure that the operation has been performed correctly, the program waits for a while. Finally the opening task is checked: if no errors are detected the signal *Clutch_placed_in_S08* is set to 1 and the robot starts moving away, out of area of ST8; otherwise an error is displayed on the Flex Pendant once the *OpeningGripper2Error* signal is set to 1. Also this time all the functions are composed by the same structure.

```
PROC place_gripper2_ST8()
  MoveL Offs(place_gripper2_S08,0.03,0.76,36.01),v1000,z50,tGripper2;
  MoveL offs(place_gripper2_S08,0,0,0),v100,fine,tGripper2;
  IF Clutch_in_gripper2 = 1 AND Gripper2_closed = 1 AND Open_gripper2 = 1 THEN
    SetDO Gripper2_opened, 1;
    SetDO Gripper2_closed, 0;
  ENDF
  WaitUntil (DInput(Clutch_in_gripper2) = 0 AND Gripper2_opened = 1)\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper2_error;
    openingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_S08, 1;
    MoveL Offs(place_gripper2_S08,0.03,0.76,36.01),v100,z50,tGripper2;
    MoveL [[570.27,72.15,1013.62],[0.00332193,0.953159,-0.302448,0.00157192],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper2;
    IF Place_NOK_C_G1 = 0 THEN
      MoveL [[299.85,590.84,1012.24],[2.82203E-05,-0.956296,0.292387,-0.00295612],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper2;
    ENDF
    openingGripper2Error := FALSE;
    place_to_ST8_G2 := TRUE;
    pick_from_ST6_G2 := FALSE;
    pick_from_ST6_G2 := FALSE;
  ENDF
ENDPROC
```

Figure 44: place task to ST8 performed by gripper 2

3.3.5. PLACE_GRIPPER2_NOK_FC_CONVEYOR

```
PROC place_gripper2_NOK_FC_conveyor()
  MoveL Offs(place_gripper2_NOK_FC,0,0,100),v1000,z50,tGripper2;
  MoveL offs(place_gripper2_NOK_FC,0,0,0),v100,fine,tGripper2;
  IF Open_gripper2 = 1 THEN
    SetDO Gripper2_opened, 1;
    SetDO Gripper2_closed, 0;
  ENDF
  WaitUntil (DInput(Clutch_in_gripper2) = 0 AND Gripper2_opened = 1)\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper2_error;
    openingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_NOK_conveyor, 1;
    MoveL Offs(place_gripper2_NOK_FC,0,0,100),v1000,fine,tGripper2;
    MoveL [[655.84,-291.38,1206.98],[0.00151811,0.937365,-0.348345,0.000577677],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tool0;
    openingGripper2Error := FALSE;
    place_to_NOK_FC_G2 := TRUE;
    pick_from_ST6_G2 := FALSE;
    pick_from_ST7_G2 := FALSE;
  ENDF
ENDPROC
```

Figure 45: place task to NOK_FC conveyor performed by gripper 2

3.3.6. PLACE_GRIPPER2_NOK_ML_CONVEYOR

```

PROC place_gripper2_NOK_ML_conveyor()
  MoveL Offs(place_gripper2_NOK_ML,0,0,100),v1000,z50,tGripper2;
  MoveL offs(place_gripper2_NOK_ML,0,0,0),v100,fine,tGripper2;
  IF Open_gripper2 = 1 THEN
    SetDO Gripper2_opened, 1;
    SetDO Gripper2_closed, 0;
  ENDIF
  WaitUntil (DInput(Clutch_in_gripper2) = 0 AND Gripper2_opened = 1)\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper2_error;
    openingGripper2Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_NOK_conveyor, 1;
    MoveL Offs(place_gripper2_NOK_ML,0,0,100),v100,z50,tGripper2;
    openingGripper2Error := FALSE;
    place_to_NOK_M_G2 := TRUE;
    pick_from_ST9_G2 := FALSE;
  ENDIF
ENDPROC

```

Figure 46: place task to NOK_ML conveyor performed by gripper 2

3.3.7. PLACE_GRIPPER1_ST9

```

PROC place_gripper1_ST9()
  MoveL [[-46.49,1207.16,1186.27],[1.19926E-06,0.105797,-0.994388,2.91914E-05],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
  MoveL [[168.55,1435.25,1086.20],[0.00446907,-0.0320222,0.999477,0.000467666],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
  MoveL Offs(place_gripper1_S09,0,0,50),v1000,z50,tGripper1;
  MoveL offs(place_gripper1_S09,0,0,0),v100,fine,tGripper1;
  IF Open_gripper1 = 1 THEN
    SetDO Gripper1_opened, 1;
    SetDO Gripper1_closed, 0;
  ENDIF
  WaitUntil (DInput(Clutch_in_gripper1) = 0 AND Gripper1_opened = 1)\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper1_error;
    openingGripper1Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_S09, 1;
    MoveL Offs(place_gripper1_S09,0,0,50),v100,z50,tGripper1;
    MoveL [[168.55,1435.25,1086.20],[0.00446907,-0.0320222,0.999477,0.000467666],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1000,z50,tGripper1;
    MoveL [[-46.49,1207.16,1186.27],[1.19926E-06,0.105797,-0.994388,2.91914E-05],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tool0;
    MoveL [[299.85,590.84,1012.24],[2.82203E-05,-0.956296,0.292387,-0.00295612],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
    openingGripper1Error := FALSE;
    place_to_ST9_G1 := TRUE;
    pick_from_ST8_G1 := FALSE;
  ENDIF
ENDPROC

```

Figure 47: place task to ST9 performed by gripper 1

3.3.8. PLACE_GRIPPER1_NOK_C_CONVEYOR

```

PROC place_gripper1_NOK_C_conveyor()
  MoveL [[121.19,-497.60,1017.33],[0.000607858,0.275829,-0.9612,0.00363941],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
  MoveL Offs(place_gripper1_NOK_C,0,0,100),v1000,z50,tGripper1;
  MoveL offs(place_gripper1_NOK_C,0,0,0),v100,fine,tGripper1;
  IF Open_gripper1 = 1 THEN
    SetDO Gripper1_opened, 1;
    SetDO Gripper1_closed, 0;
  ENDIF
  WaitUntil (DInput(Clutch_in_gripper1) = 0 AND Gripper1_opened = 1)\MaxTime:=1\TimeFlag:=bTimeOut;
  IF bTimeOut THEN
    !Error
    SetDO Error_signal, 1;
    TPErase;
    TPWrite opening_gripper1_error;
    openingGripper1Error := TRUE;
    RETURN;
  ELSE
    SetDO Clutch_placed_in_NOK_conveyor, 1;
    MoveL Offs(place_gripper1_NOK_C,0,0,100),v100,z50,tGripper1;
    MoveL [[121.19,-497.60,1017.33],[0.000607858,0.275829,-0.9612,0.00363941],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tGripper2;
    openingGripper1Error := FALSE;
    place_to_NOK_C_G1 := TRUE;
    pick_from_ST8_G1 := FALSE;
  ENDIF
ENDPROC

```

!erase on the Flex pendant the previous message
!show the error
!set the error flag

!clutch placed in NOK_C conveyor
!go away from clutch

Figure 48: place task to NOK_C conveyor performed by gripper 1

4. MOVEMENTS CONFIGURATION

To guarantee safety inside plants where robots are not completely isolated, but interfere with machines (used for example for pick and place tasks), ABB provides an hardware functionality called SafeMove option. It is integrated thanks to a safety fieldbus connected to the IRC5 controller and guarantees that in whatever situation, robot will never overcome the edges defined through RobotStudio visualizer.

It also eliminates the need for an external safety PLC by incorporating safe fieldbus communications directly into the robot controller. SafeMove still includes dedicated hardware to ensure the performance of the safety system, including a reliable safety I/O which guarantees that the application running on the controller's main computer operates independently and particularly in a predictable way.

So, given the shape correctly dimensioned of the area in which the robot can move without causing any damages and eventually adding some allowed tolerances, the software is able to determine the joints variables necessary to reduce the movements ranges.

In addition, supervisor options can be included to monitor the tool speed, position and orientation plus axis position and speed. However, it is necessary to consider the fact that SafeMove limits represents the points where the robot starts to decelerate, so it is necessary to set up them a little bit before in order to avoid undesirable crashes against the gates or the perimetral structure of the line.

If robots are not equipped with SafeMove option, something similar can be achieved by configuring WorldZone by software, inside the same Rapid section where program is written. Particularly, since all of them provide this functionality, I used it to set the *home_position* flag and all the others related to the encumbrance of a certain station. Moreover, I imposed a superior limit for R2 to be sure that it will never interfere with the arms of manipulator M1 in high position and with the machines' mechanical structures.

However, a distinction needs to be done to differentiate the management of the home position signal with respect to the one related to the encumbrances. In fact, in practice the home position is represented by a single point, which corresponds to a specific configuration of the six axis, so to a certain angular position for each of them. Due to this fact, the robot is no more in home position if it moves far from it of a certain tolerance that can be set for each axis. So, to guarantee a good precision and to improve the control that the PLC can do on the robot, I decided to set a tolerance of 1° for each axis. This means that even for very small movements, performed for example in manual mode, the robot signals to the PLC that it is no more in home position. For this purpose, I declared it as a *jointtarget* variable, which differs from the normal *robtarg* since it is expressed in degrees for each joint and not in coordinate form.

All the encumbrances, instead, are defined as areas characterized by the fact that if the robot is positioned inside one of them, then there could be damages or problems in case some other elements of the environment are moved. This means that they act as limitations for the normal working situation of the whole line. To this purpose, they have been declared not as single points as happened for the home position, but as volumes cubical, cylindrical or shaped like a parallelepiped, using obviously the tool which interacts with that zone. Every time the tool enters that zone, the encumbrance signal is set to 0 to inform the PLC to stop every movements of the machines. In addition, all the signals that will be used inside the WZones must be set as ReadOnly signals, to be sure that they won't be overwritten manually, but only by the WZones themselves. Then, the default value of the encumbrance signals, has been changed into 1 to indicate that the approach to the machines is not continuous.

In general, the most critical situations involve the manipulator area (ST2 and ST5 stations), the two machines (ST6, ST7 and ST8 stations) and finally the marking area (ST9 station). Particularly, for the manipulator area, there are limitation concerning the rotation of the manipulator itself and of the rotating

equipment which is positioned on the station to be able to fit the clutch mechanism which is placed there. Every time a new clutch is placed inside ST5, the manipulator goes down, closes its gripper and pick the item. Then it returns to the high position and rotates of one step (since there are 5 steps, it does a rotation of 72° in counterclockwise direction). In this situation encumbrance is necessary to assure that the manipulator goes down and then starts to rotate only when the robot is far from both ST2 and ST5 simultaneously, otherwise it will crash against the robot. For the machines, instead, it is necessary to analyze the two situation separately. For what concerns finger correction machine, it is provided by a sort of drawer which has been introduced to gain cycle time. Since it is feed and discharged alternatively in ST6 and ST7, every time a new part has to be placed in one of the two stations, the drawer has to be unlocked by the obturators used to fix it, then it has to move up to get in touch with the guides and finally it can translate to the other position. Once it has completed its movement, it must go down again to be fixed by the obturators in the other position. Since this operation requires not only a translation, but also a vertical movement, it is necessary that the robot is still far from it while it is moving, avoiding to crash. The functional control machine, instead, provides a single station ST8, which is located exactly below the main block that, during the test, moves down to get in touch with the mechanism and performs the simulation of the behaviour of the flywheel inside the vehicle. So, since the robot approach the machine until its gripper comes "inside" it, it is very important that the movement along the vertical direction of the machine is done only when the robot is out of its area. Finally, in ST9 there is a rotating table composed of 5 stations, where mechanisms has to be placed to be marked by two laser station and then with the application of a label containing a DataMatrix. Then, if they are ok, they are moved to ST13 and moved away from the line by the unloading manipulator, otherwise they are picked up again by robot R3 and placed inside the NOK_ML conveyor. All the stations of the rotating table are divided by alluminium profiles which are fixed radially with respect to the center of the table, so the encumbrance area of ST9 starts before the table and ends almost in correspondance of its center.

To impose software limits on the vertical position that are valid also when manual movements are performed, I added a volume with almost the same dimensions of the encumbrance areas, but with the main difference related to the fact that its minimum height is the maximum one that can be reached by the robot arm, while the upper one theoretically should be infinite, but I set it to 6 meters so that I am sure the robot will never be able to overcome it. The limits have been declared for all the stations where the robots has to pick or place something, even if they are important particularly near the manipulator (ST2 and ST5) and inside the functional control machine (ST8).

4.1. ROBOT R1

4.1.1. SAFEMOVE

Starting from the layout representation, I configured the SafeMove volume in which theoretically robot R1 is allowed to move. Then I reduced it to be sure that the robot was able to stop in time without crashing against the line protections. Finally I imposed an height from the ground of 1500 mm.

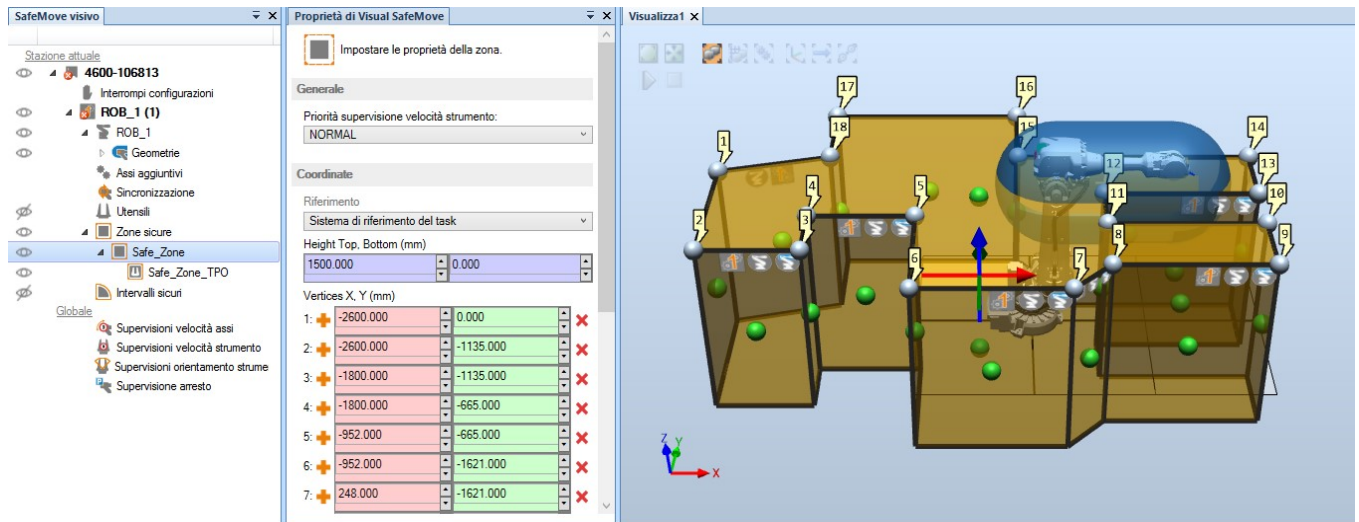


Figure 49: SafeMove configuration of robot R1

4.1.2. WZONES AND LIMITS

Name	Type of Signal	Assigned to Device	Signal Identification Label	Devic	Categori	Access Level	Default Value
Out_of_area_entry_line	Digital Output	PN_Internal_Device		40		ReadOnly	1
Out_of_area_NOK_conveyor	Digital Output	PN_Internal_Device		42		ReadOnly	1
Out_of_area_S05	Digital Output	PN_Internal_Device		41		ReadOnly	1
Out_of_area_S06	Digital Output	PN_Internal_Device		43		ReadOnly	1
Out_of_area_S07	Digital Output	PN_Internal_Device		44		ReadOnly	1

Figure 50: signals configuration for robot R1

```
VAR shapedata volRobotHome;
VAR wzstationary wz_RobotHome;
PERS jointtarget home_pos:=[[[-115.45,7.31925,14.9134,-0.697995,68.4028,64.5194],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]];
CONST jointtarget delta_pos:=[[[1,1,1,1,1,1],[9E9,9E9,9E9,9E9,9E9,9E9]]];
```

Figure 51: home position declaration for robot R1

```
! HOME POS
home_pos:=CalcJointT(homePos,Tool:=tool0\WObj:=wobj0);
WZHomeJointDef\Inside,volRobotHome,home_pos,delta_pos;
WZDOSet\Stat,wz_RobotHome\Inside,volRobotHome,Robot_in_home_pos,1;
```

Figure 52: home position for robot R1

```

! Box
VAR shapedata volFIngST6;
VAR wzstationary wz_FIngST6;
PERS pos p_C1_IngST6:=[-526,1102,1340];!from entry point
PERS pos p_C2_IngST6:=[-200,1650,995];!to place point

! Box
VAR shapedata volFIngST7;
VAR wzstationary wz_FIngST7;
PERS pos p_C1_IngST7:=[560.51,1100.67,1083.84];!from entry point
PERS pos p_C2_IngST7:=[620.51,1650.67,983.84];!to place point

! Box
VAR shapedata volFIngSTA;
VAR wzstationary wz_FIngSTA;
PERS pos p_C1_IngSTA:=[-575.53,-1651.73,930];!from entry point
PERS pos p_C2_IngSTA:=[-640.53,-990.73,1050];!to pick point

! Box
VAR shapedata volFIngST5;
VAR wzstationary wz_FIngST5;
PERS pos p_C1_IngST5:=[1050.07,-370.60,990.43];!from entry point
PERS pos p_C2_IngST5:=[960.07,-410.60,1050.43];!to pick/place point

! Box
VAR shapedata volFIngNOK_A_conv;
VAR wzstationary wz_FIngNOK_A_conv;
PERS pos p_C1_NOK_A_conv:=[0,-1651,750];!from entry point
PERS pos p_C2_NOK_A_conv:=[-500,-990,-500];!to place point

! Box
VAR shapedata volFIngNOK_B_conv;
VAR wzstationary wz_FIngNOK_B_conv;
PERS pos p_C1_NOK_B_conv:=[0,-1651,1250];!from entry point
PERS pos p_C2_NOK_B_conv:=[-500,-990,760];!to place point

```

Figure 53: WZones definition for robot R1

```

! BOX
WZBoxDef\Inside,volFIngST6,p_C1_IngST6,p_C2_IngST6;
WZDOSet\Stat,wz_FIngST6\Before,volFIngST6,Out_of_area_S06,0;

! BOX
WZBoxDef\Inside,volFIngST7,p_C1_IngST7,p_C2_IngST7;
WZDOSet\Stat,wz_FIngST7\Before,volFIngST7,Out_of_area_S07,0;

! BOX
WZBoxDef\Inside,volFIngSTA,p_C1_IngSTA,p_C2_IngSTA;
WZDOSet\Stat,wz_FIngSTA\Before,volFIngSTA,Out_of_area_entry_line,0;

! BOX
WZBoxDef\Inside,volFIngST5,p_C1_IngST5,p_C2_IngST5;
WZDOSet\Stat,wz_FIngST5\Before,volFIngST5,Out_of_area_S05,0;

! BOX
WZBoxDef\Inside,volFIngNOK_A_conv,p_C1_NOK_A_conv,p_C2_NOK_A_conv;
WZDOSet\Stat,wz_FIngNOK_A_conv\Before,volFIngNOK_A_conv,Out_of_area_NOK_conveyor,0;

! BOX
WZBoxDef\Inside,volFIngNOK_B_conv,p_C1_NOK_B_conv,p_C2_NOK_B_conv;
WZDOSet\Stat,wz_FIngNOK_B_conv\Before,volFIngNOK_B_conv,Out_of_area_NOK_conveyor,0;

```

Figure 54: WZones for robot R1

```

! LIMIT
VAR shapedata VollimtSTA;
VAR wzstationary WZ_LimitSTA:=[];
PERS pos C1_STA:=[-575.53,-1651.73,960];
PERS pos C2_STA:=[-640.53,-990.73,6000];

! LIMIT
VAR shapedata VollimtST5;
VAR wzstationary WZ_LimitST5:=[];
PERS pos C1_ST5:=[1050.07,-370.60,1100];
PERS pos C2_ST5:=[960.07,-410.60,6000];

! LIMIT
VAR shapedata VollimtST6;
VAR wzstationary WZ_LimitST6:=[];
PERS pos C1_ST6:=[-526,1102,1360];
PERS pos C2_ST6:=[-200,1650,6000];

! LIMIT
VAR shapedata VollimtST7;
VAR wzstationary WZ_LimitST7:=[];
PERS pos C1_ST7:=[560.51,1100.67,1100.84];
PERS pos C2_ST7:=[620,1650,6000];

! LIMIT
VAR shapedata VollimtNOK1_A;
VAR wzstationary WZ_LimitNOK1_A:=[];
PERS pos C1_NOK1_A:=[0,-1651,800];
PERS pos C2_NOK1_A:=[-500,-990,800];

! LIMIT
VAR shapedata VollimtNOK1_B;
VAR wzstationary WZ_LimitNOK1_B:=[];
PERS pos C1_NOK1_B:=[0,-1651,1500];
PERS pos C2_NOK1_B:=[-500,-990,6000];

```

Figure 55: limits definition for robot R1

```

! LIMITE
WZBoxDef\Inside,VollimtSTA,C1_STA,C2_STA;
WZLimSup\Stat,WZ_LimitSTA,VollimtSTA;

! LIMITE
WZBoxDef\Inside,VollimtST5,C1_ST5,C2_ST5;
WZLimSup\Stat,WZ_LimitST5,VollimtST5;

! LIMITE
WZBoxDef\Inside,VollimtST6,C1_ST6,C2_ST6;
WZLimSup\Stat,WZ_LimitST6,VollimtST6;

! LIMITE
WZBoxDef\Inside,VollimtST7,C1_ST7,C2_ST7;
WZLimSup\Stat,WZ_LimitST7,VollimtST7;

! LIMITE
WZBoxDef\Inside,VollimtNOK1_A,C1_NOK1_A,C2_NOK1_A;
WZLimSup\Stat,WZ_LimitNOK1_A,VollimtNOK1_A;

! LIMITE
WZBoxDef\Inside,VollimtNOK1_B,C1_NOK1_B,C2_NOK1_B;
WZLimSup\Stat,WZ_LimitNOK1_B,VollimtNOK1_B;

```

Figure 56: limits for robot R1

4.2. ROBOT R2

4.2.1. WZONES AND LIMITS

Name	Assigned to Device	Type of Signal	Device Mapping	Si	Category	Access Level	Default Value
Out_of_ST2	PN_Internal_Device	Digital Output	40			ReadOnly	1
Out_of_rivet_feeder	PN_Internal_Device	Digital Output	41			ReadOnly	1

Figure 57: signals definition definition for robot R2

```
! HOME POS
home_pos:=CalcJointT(homePos,Tool:=tool0\WObj:=wobj0);
WZHomeJointDef\Inside,volRobotHome,home_pos,delta_pos;
WZDOSet\Stat,wz_RobotHome\Inside,volRobotHome,Robot_in_home_pos,1;
```

Figure 58: home position definition for robot R2

```
VAR shapedata volRobotHome;
VAR wzstationary wz_RobotHome;
PERS jointtarget home_pos:=[[1.21831,-27.0864,21.5705,-0.000122169,95.5652,-21.6269],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]];
CONST jointtarget delta_pos:=[[1,1,1,1,1,1],[9E9,9E9,9E9,9E9,9E9,9E9]];
```

Figure 59: home position for robot R2

```
! Cylinder
VAR shapedata volFIngFeeder;
VAR wzstationary wz_FIngFeeder;
PERS pos p_C1_Ing2:=[-1000,-423,750];
PERS pos p_C2_Ing2:=[750,-180,-500];

! Box
VAR shapedata volFIngSt2;
VAR wzstationary wz_FIng2FIngSt2;
CONST pos CIng_St2:=[-340,396,-500];
CONST num RIST2:=260;
CONST num HIngSt2:=1150;
```

Figure 60: WZones definition for robot R2

```

! Cylinder
WZCylDef\Inside,volFIngSt2,CIng_St2,RIS2,HIngSt2;
WZDOSet\Stat,wz_FIng2FIngSt2\Before,volFIngSt2,Out_of_ST2,0;

! Box
WZBoxDef\Outside,volFIngFeeder,p_C1_Ing2,p_C2_Ing2;
WZDOSet\Stat,wz_FIngFeeder\Before,volFIngFeeder,Out_of_rivet_feeder,1;

```

Figure 61: WZones for robot R2

```

! LIMIT
VAR shapedata Vollimt;
VAR wzstationary WZ_Limit:=[0];
PERS pos C1_BOX:=[-1000,-380,820];
PERS pos C2_BOX:=[150,615,6000];

```

Figure 62: limit definition for robot R2

```

! LIMIT
WZBoxDef\Inside,Vollimt,C1_BOX,C2_BOX;
WZLimSup\Stat,WZ_Limit,Vollimt;

```

Figure 63: limit for robot R2

4.3. ROBOT R3

4.3.1. SAFEMOVE

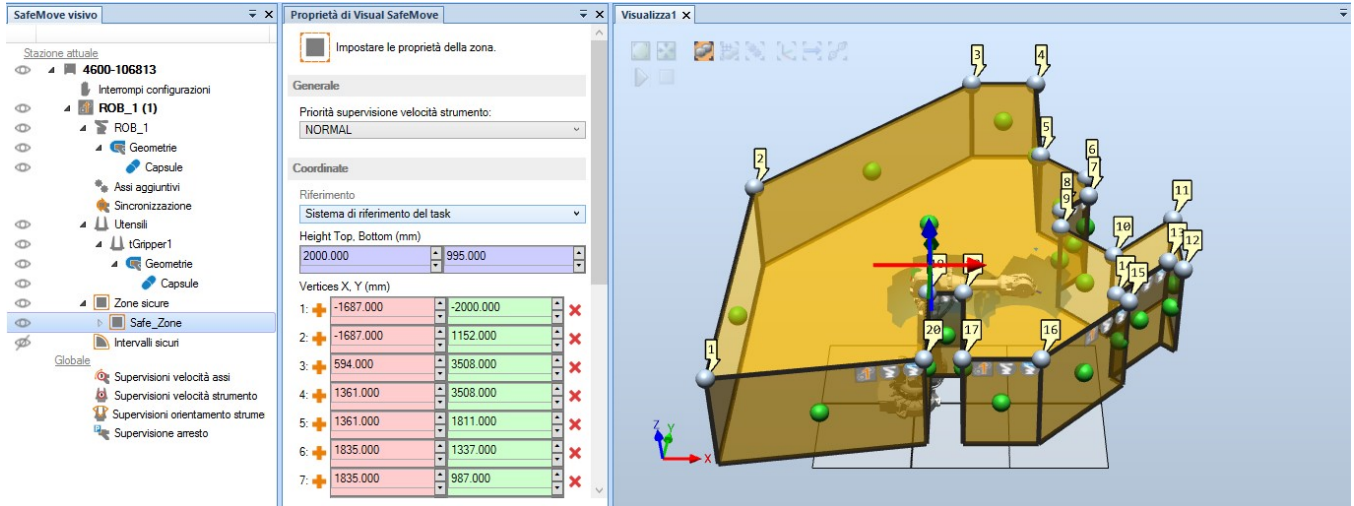


Figure 64: SafeMove configuration for robot R3

4.3.2. WZONES AND LIMITS

Name	Assigned to Device	Type of Signal	Device Mapping	Si	Category	Access Level	Default Value
Robot_in_home_pos	PN_Internal_Device	Digital Output	26			ReadOnly	0
Clutch_picked_from_S06	PN_Internal_Device	Digital Output	32			Default	0
Clutch_picked_from_S07	PN_Internal_Device	Digital Output	33			Default	0
Clutch_picked_from_S08	PN_Internal_Device	Digital Output	34			Default	0
Clutch_placed_in_S08	PN_Internal_Device	Digital Output	35			Default	0
Clutch_placed_in_S09	PN_Internal_Device	Digital Output	36			Default	0
Clutch_NOK_picked_from_S09	PN_Internal_Device	Digital Output	37			Default	0
Clutch_placed_in_NOK_conveyor	PN_Internal_Device	Digital Output	38			Default	0
Out_of_area_S06	PN_Internal_Device	Digital Output	40			ReadOnly	1
Out_of_area_S07	PN_Internal_Device	Digital Output	41			ReadOnly	1
Out_of_area_S08	PN_Internal_Device	Digital Output	42			ReadOnly	1
Out_of_area_S09	PN_Internal_Device	Digital Output	43			ReadOnly	1
Out_of_area_NOK_conveyor	PN_Internal_Device	Digital Output	44			ReadOnly	1

Figure 65: signals configuration for robot R3

```

VAR shapedata volRobotHome;
VAR wzstationary wz_RobotHome;
PERS jointtarget home_pos:= [[-86.0272, -42.8516, 39.0619, -0.520661, 93.791, -266.552], [9E+9, 9E+9, 9E+9, 9E+9, 9E+9, 9E+9]];
CONST jointtarget delta_pos:= [[1, 1, 1, 1, 1, 1], [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];

```

Figure 66: home position definition for robot R3

```

! HOME POS
home_pos:=CalcJointT(homePos, Tool:=tool0\Wobj:=wobj0);
WZHomeJointDef\Inside, volRobotHome, home_pos, delta_pos;
WZDOSet\Stat, wz_RobotHome\Inside, volRobotHome, Robot_in_home_pos, 1;

```

Figure 67: home position for robot R3

```

! Box
VAR shapedata volFIngST6;
VAR wzstationary wz_FIngST6;
PERS pos p_C1_IngST6:=[-253,-1650,1120];
PERS pos p_C2_IngST6:=[-560,-1096,995];

! Box
VAR shapedata volFIngST7;
VAR wzstationary wz_FIngST7;
PERS pos p_C1_IngST7:=[298,-1103,1165];
PERS pos p_C2_IngST7:=[1018,-1540,995];

! Box
VAR shapedata volFIngST8;
VAR wzstationary wz_FIngST8;
PERS pos p_C1_IngST8:=[1214,-317,995];
PERS pos p_C2_IngST8:=[1550,-700,1033];

! Box
VAR shapedata volFIngST9;
VAR wzstationary wz_FIngST9;
PERS pos p_C1_IngST9:=[440,1617,1070];
PERS pos p_C2_IngST9:=[769,1384,995];

! Box
VAR shapedata volFIngNOK_C_conv;
VAR wzstationary wz_FIngNOK_C_conv;
PERS pos p_C1_NOK_C_conv:=[1500,-15,990];
PERS pos p_C2_NOK_C_conv:=[1700,0,1100];

! Box
VAR shapedata volFIngNOK_FC_conv;
VAR wzstationary wz_FIngNOK_FC_conv;
PERS pos p_C1_NOK_FC_conv:=[1250.39,75,1000.73];
PERS pos p_C2_NOK_FC_conv:=[1700,110,1500];

! Box
VAR shapedata volFIngNOK_M_conv;
VAR wzstationary wz_FIngNOK_M_conv;
PERS pos p_C1_NOK_M_conv:=[1300,40,900];
PERS pos p_C2_NOK_M_conv:=[1700,85,1200];

```

Figure 68: WZones definition for robot R3


```

! BOX
WZBoxDef\Inside,volFIngST6,p_C1_IngST6,p_C2_IngST6;
WZDSet\Stat,wz_FIngST6\Before,volFIngST6,Out_of_area_S06,0;

! BOX
WZBoxDef\Inside,volFIngST7,p_C1_IngST7,p_C2_IngST7;
WZDSet\Stat,wz_FIngST7\Before,volFIngST7,Out_of_area_S07,0;

! BOX
WZBoxDef\Inside,volFIngST8,p_C1_IngST8,p_C2_IngST8;
WZDSet\Stat,wz_FIngST8\Before,volFIngST8,Out_of_area_S08,0;

! BOX
WZBoxDef\Inside,volFIngST9,p_C1_IngST9,p_C2_IngST9;
WZDSet\Stat,wz_FIngST9\Before,volFIngST9,Out_of_area_S09,0;

! BOX
WZBoxDef\Outside,volFIngNOK_C_conv,p_C1_NOK_C_conv,p_C2_NOK_C_conv;
WZDSet\Stat,wz_FIngNOK_C_conv\Before,volFIngNOK_C_conv,Out_of_area_NOK_conveyor,0;

! BOX
WZBoxDef\Outside,volFIngNOK_FC_conv,p_C1_NOK_FC_conv,p_C2_NOK_FC_conv;
WZDSet\Stat,wz_FIngNOK_FC_conv\Before,volFIngNOK_FC_conv,Out_of_area_NOK_conveyor,0;

! BOX
WZBoxDef\Outside,volFIngNOK_M_conv,p_C1_NOK_M_conv,p_C2_NOK_M_conv;
WZDSet\Stat,wz_FIngNOK_M_conv\Before,volFIngNOK_M_conv,Out_of_area_NOK_conveyor,0;

```

Figure 69: WZones for robot R3

```

! LIMIT
VAR shapedata VolLimit_NOK_C;
VAR wzstationary WZ_Limit_NOK_C:=[];
PERS pos C1_BOX_NOK_C:=[1500,-15,1100];
PERS pos C2_BOX_NOK_C:=[1700,0,6000];

! LIMIT
VAR shapedata VolLimit_NOK_FC;
VAR wzstationary WZ_Limit_NOK_FC:=[];
PERS pos C1_BOX_NOK_FC:=[1250.39,75,1100];
PERS pos C2_BOX_NOK_FC:=[1700,110,6000];

! LIMIT
VAR shapedata VolLimit_NOK_M;
VAR wzstationary WZ_Limit_NOK_M:=[];
PERS pos C1_BOX_NOK_M:=[1300,40,1000];
PERS pos C2_BOX_NOK_M:=[769,1384,6000];

```

Figure 70: limits definition for robot R3


```
! LIMIT
WZBoxDef\Inside,Vollimt_ST6,C1_BOX_ST6,C2_BOX_ST6;
WZlimSup\Stat,WZ_Limit_ST6,Vollimt_ST6;

! LIMIT
WZBoxDef\Inside,Vollimt_ST7,C1_BOX_ST7,C2_BOX_ST7;
WZlimSup\Stat,WZ_Limit_ST7,Vollimt_ST7;

! LIMIT
WZBoxDef\Inside,Vollimt_ST8,C1_BOX_ST8,C2_BOX_ST8;
WZlimSup\Stat,WZ_Limit_ST8,Vollimt_ST8;

! LIMIT
WZBoxDef\Inside,Vollimt_ST9,C1_BOX_ST9,C2_BOX_ST9;
WZlimSup\Stat,WZ_Limit_ST9,Vollimt_ST9;

! LIMIT
WZBoxDef\Inside,Vollimt_NOK_C,C1_BOX_NOK_C,C2_BOX_NOK_C;
WZlimSup\Stat,WZ_Limit_NOK_C,Vollimt_NOK_C;

! LIMIT
WZBoxDef\Inside,Vollimt_NOK_FC,C1_BOX_NOK_FC,C2_BOX_NOK_FC;
WZlimSup\Stat,WZ_Limit_NOK_FC,Vollimt_NOK_FC;

! LIMIT
WZBoxDef\Inside,Vollimt_NOK_M,C1_BOX_NOK_M,C2_BOX_NOK_M;
WZlimSup\Stat,WZ_Limit_NOK_M,Vollimt_NOK_M;
```

Figure 71: limits for robot R3

5. CONCLUSIONS

The main difficulties I found during this work are related particularly to the management of the movements. The first problem is connected to the grippers: except for robot R2 which is smaller with respect to the other two, I had to manage two grippers of more than 30 kg and of a maximum length of 710 mm and this was not easy for many reasons. First of all the weight was important and move, particularly in manual mode, in a narrow environment as that one was, increased the risk of crashing against something causing potentially big damages both to the grippers and to the machines.

Then there was the problem related to the physical connections of the grippers to the robot arm. Since the gripper is composed by two tools which are treated separately, each of them is equipped with an hydraulic cylinder which, thanks to the presence or absence of the air, moves inside its chamber transmitting its movements to the mechanical plugs that are used to keep the clutch mechanism, so causing the opening or the closing of the tool itself, together with the presence of electrovalves able to regulate the air flux to and from the cylinder. In addition to the mechanical equipment, it is necessary to consider also the presence of some photocells that are used to signal the gripper state (if opened or closed) and the presence of a mechanism inside it. So, looking at the complete system, the movements in general and in particular the rotations around the TCP have been performed so that cables returned every time to a certain position without running the risk of pull them too much, creating problems to the electronic and mechanical devices. This was possible also considering the information that are included inside the *robtarg* parameters, immediately after the position coordinates and the joints' orientation in quaternion form. Particularly this vector is composed by 4 elements: the first indicates the forward or backward orientation of the robot, the second the up or down orientation of axis 4, the third the positive or negative orientation of axis 6 while the last is related to the type of robot, in our case indicates the position of axis 2. It works as follows:

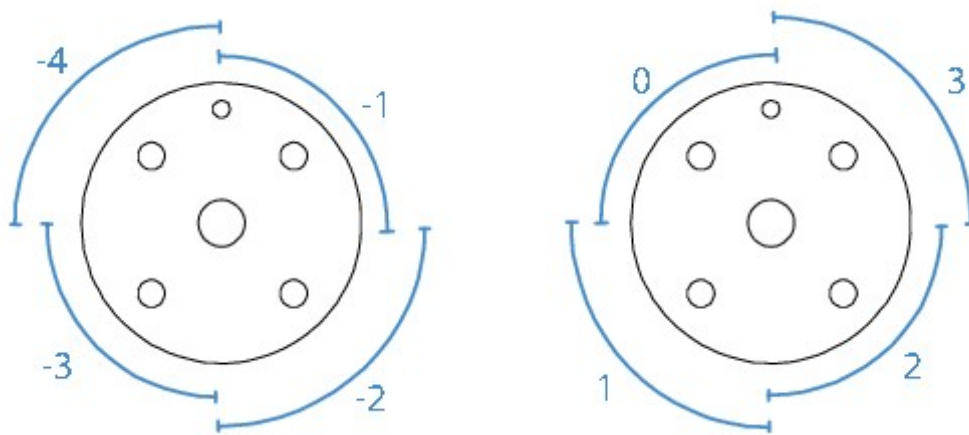


Figure 72: configuration parameters of robtarget

Togheter with the above considerations, the other problem was related to the orientation of the clutch mechanism which was requested to be constant during all the steps inside the line. Starting from the fact that each component can be divided in three sectors, each of 120°, for the pick and place tasks it was necessary for the mechanical plugs of the gripper, to be as closer as possible to the centering pins obtaining advantages also from the point of view of the accuracy with which movements have been defined. This way, it was possible to keep constant the orientato of the DataMatrix too.

From a more general point of view, instead, the two main difficulties are related to the environment disposition of all the line components and by the limited cycle time. This was a situation in which the robot has to rotate many times the tool to first discharge and then charge again a certain station, without the need of performing other operations in between. However, since all the stations were located in narrow places with not sufficient free space around to be able to do the rotation of the tool without moving completely away from the work area, cycle time increases.

To try to compensate the delay caused by movements that have been introduced just to avoid collision while the gripper was reoriented, I choosed always linear movements. This way the trajectories were inherently optimized beacuse the robot moved along the shortest possible path. Morover, all the intermidate points that were necessary to introduce to force the robot to move far from the machines, have been defined as much as possible along the path the robot would have done in absence of any obstacles, so imposing just translation along a certain direction.

Finally, a big advantage is constituted by the safety options that ABB provides, both in software and hardware, that limit some types of movement, increasing the reliability of the whole system.

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor Prof. M. Violante for his availability, the company Aurotec engineering s.a.s and all my colleagues for helping me.

I would like to thank my parents, my brother Marco and Nicola for supporting me every day.

I would like to thank all my family, my grandmother Francesca and my uncle Piero.

Finally thanks to my classmates and to all my friends.

BIBLIOGRAPHY:

1. Riad Menasri, Hamouche Oulhadj, Boubaker Daachi, Amir Nakib and Patrick Siarry, *A genetic algorithm designed for robot trajectory planning*, 2014 IEEE International Conference on Systems, Man, and Cybernetics October 5-8, 2014, San Diego, CA, USA
2. Basilio Bona, *Modellistica dei robot industriali*, CELID, Torino 2002
3. B. Siciliano, L. Sciacicco, L. Villani, G. Oriolo, *Robotics: modelling, planning and control*, Springer-Verlag, 2009
4. <https://www.robots.com/articles/what-is-roboguide>
5. RobotStudio guides