Robotics Project Report

“Industrial and Collaborative Robotics” – Digital Automation Engineering

Andrea Prozzo – nr 205998 – 259473@studenti.unimore.it

1. Introduction

This report presents a collection of three distinct projects, each focusing on a different robotics application and environment. The first part involves generating collision-free trajectories for a robotic manipulator (UR5) to perform a pick-and-place operation. The goal is to move the robot from an initial to a final configuration in joint space while avoiding obstacles, using motion planning and trajectory generation techniques. The second part focuses on designing a controller for a differential drive mobile robot performing a lawn-mowing task. The objective here is to develop a trajectory tracking controller that ensures the robot accurately follows a predefined path across a planar area. The implementation of these first two parts is carried out in MATLAB, using the Robotics Toolbox and a set of pre-defined functions provided as part of the course material. The third and final part explores the use of ABB RobotStudio to simulate a collaborative pick-and-place task performed by an industrial manipulator operating in Speed and Separation Monitoring (SSM) mode. The robot rearranges objects between workstations while dynamically adjusting its behavior in response to human presence, using sensorized safety zones.

2. Kinematic Manipulator

In this first part of the project, a UR5 manipulator is taken into account. The initial task consists in generating a joint-space trajectory that brings the robot from a given start configuration (with zero initial velocity) to a specified goal configuration (also with zero final velocity), in a total time of 2 seconds. The vectors defining the joint variables for the start and goal configurations are specified below.

$$q(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \dot{q}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad q(2) = \begin{bmatrix} c_1/2\pi \\ c_2/2\pi \\ c_3/2\pi \\ c_4/2\pi \\ c_5/2\pi \\ c_1/2\pi \end{bmatrix} \quad \dot{q}(2) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Where parameters c_i are obtained from the identification number, set respectively as $c_1 = 2, c_2 = 5, c_3 = 8, c_4 = 9, c_5 = 9$. For this first task the approach consists in using a third-order polynomial trajectory that interpolates between the initial and goal configurations. Since the task takes place in an obstacle-free environment, no motion planning algorithms are used, the path just consist in the initial and final configuration. The state validator of the robot is only used to check the feasibility of the goal configuration through a simple validation loop. Since acceleration boundary conditions cannot be imposed through a third-order polynomial, a discontinuity in the acceleration profile is expected. Below are some key excerpts from the MATLAB code developed for this first task.

```
t0 = 0;
tf = 2;
t = t0:0.01:tf;   %Time vector
n_joints = 6;

%Initialization of pos,vel,acc for each joint
q_t = zeros(n_joints, length(t));
dq_t = zeros(n_joints, length(t));
ddq_t = zeros(n_joints, length(t));

for i = 1:n_joints
    q0 = startConfiguration(i);
    qf = goalConfiguration(i);
    dq0 = 0;  % Initial and final velocity equal to zero
    dqf = 0;

    % Cubic coefficients
    a0 = q0;
    a1 = 0;
    a2 = 3*(qf - q0)/tf^2;
    a3 = 2*(q0 - qf)/tf^3;

    % Trajectories
    q_t(i, :) = a0 + a1*t + a2*t.^2 + a3*t.^3;
    dq_t(i, :) = a1 + 2*a2*t + 3*a3*t.^2;
    ddq_t(i, :) = 2*a2 + 6*a3*t;
end
```

A time vector over which the trajectory will be evaluated is defined, spanning from 0 to 2 seconds with a sampling interval of 0.01 seconds. Matrices are then initialized to store the position, velocity, and acceleration values for all six joints at each time step. Within a loop that iterates over each joint, the code retrieves the initial and final joint positions. Since both the initial and final joint velocities are assumed to be zero, the trajectory can be described using a cubic polynomial that satisfies these boundary conditions. The coefficients of the polynomial are computed analytically and once they are determined, the position, velocity, and acceleration profiles for each joint are calculated using the standard formulas for a cubic polynomial and its derivatives.

A third order polynomial trajectory is obtained for each one of the joints. The plots below show the results in term of joint position, velocities and acceleration. The final joints position are successfully reached with zero final velocities. The total duration of the trajectory is 2 seconds, as requested.

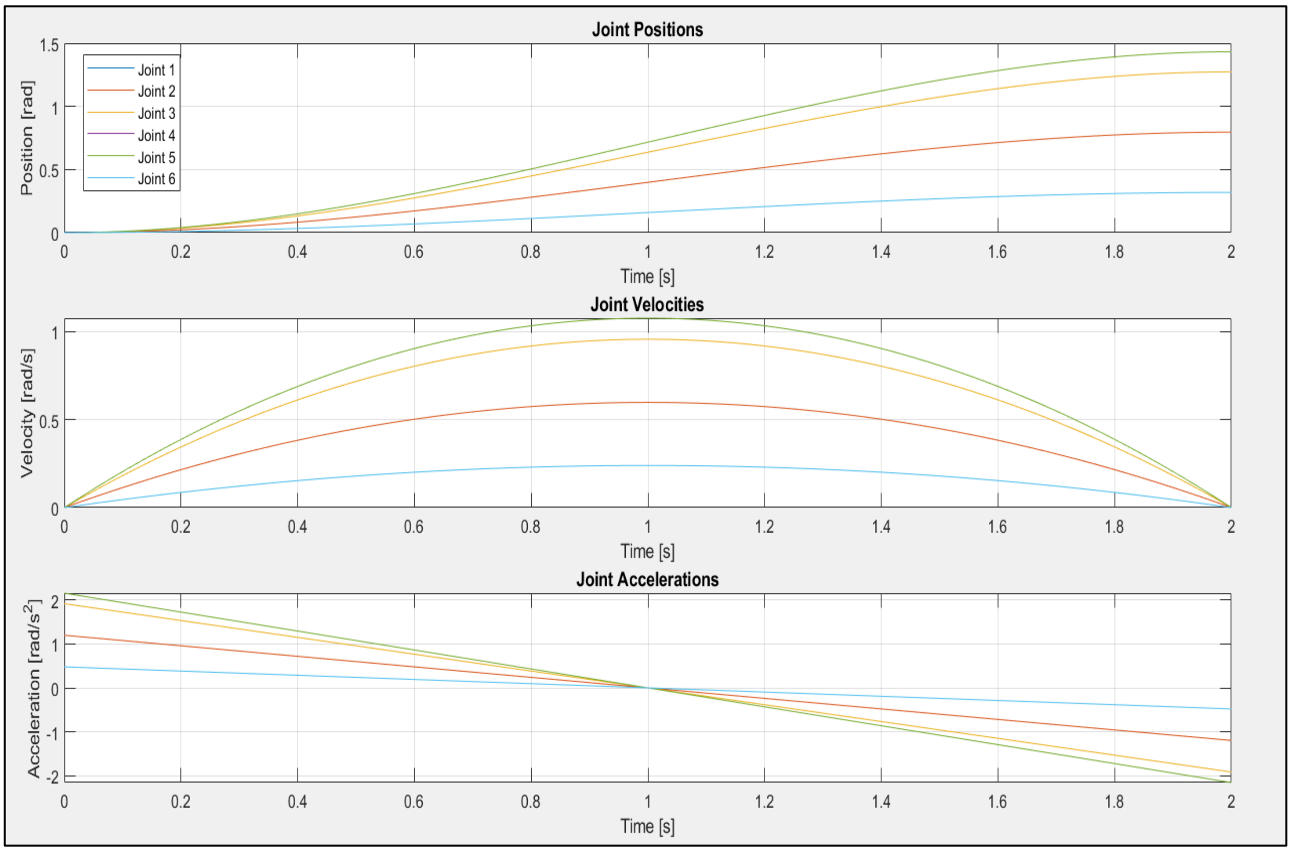
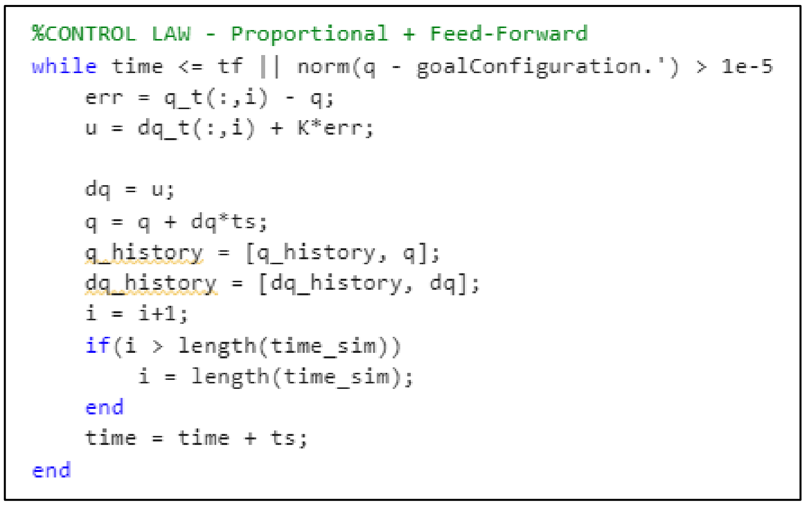


Figure 1. Third order trajectory

As expected the acceleration profiles present initial and final discontinuities that can be easily solved using a fifth-order polynomial, being able to impose boundary condition on acceleration too.

Only for the first task, a controller that effectively produces the control effort required to allow the robot to track the defined cubic trajectory is built. The objective is to implement a simple proportional controller for trajectory tracking in joint space.



```
%CONTROL LAW - Proportional + Feed-Forward
while time <= tf || norm(q - goalConfiguration.') > 1e-5
    err = q_t(:,i) - q;
    u = dq_t(:,i) + K*err;

    dq = u;
    q = q + dq*ts;
    q_history = [q_history, q];
    dq_history = [dq_history, dq];
    i = i+1;
    if(i > length(time_sim))
        i = length(time_sim);
    end
    time = time + ts;
end
```

The control gain K is defined as a 6×6 identity matrix, meaning that each joint is independently controlled using a proportional feedback loop. A history of the joint positions over time is initialized to monitor the evolution of the system. The control loop runs until either the simulation time exceeds the final time or the joint configuration reaches the goal configuration within a small tolerance. At each iteration, the tracking error is computed as the difference between the desired joint position (taken from the precomputed trajectory q_t) and the current joint position q. The control input u is then calculated by adding a feedforward term (the desired joint velocity dq_t) in order to obtain a perfect tracking, even when the gain K is low. The joint velocities dq are updated using the control input, and the joint positions are incremented using a simple integration. The updated joint positions and joint velocities are stored in q_history and dq_history for visualization. The plots below show the almost perfect overlap between the desired trajectory (dashed line) and the actual trajectory tracked with the implementation of the controller.

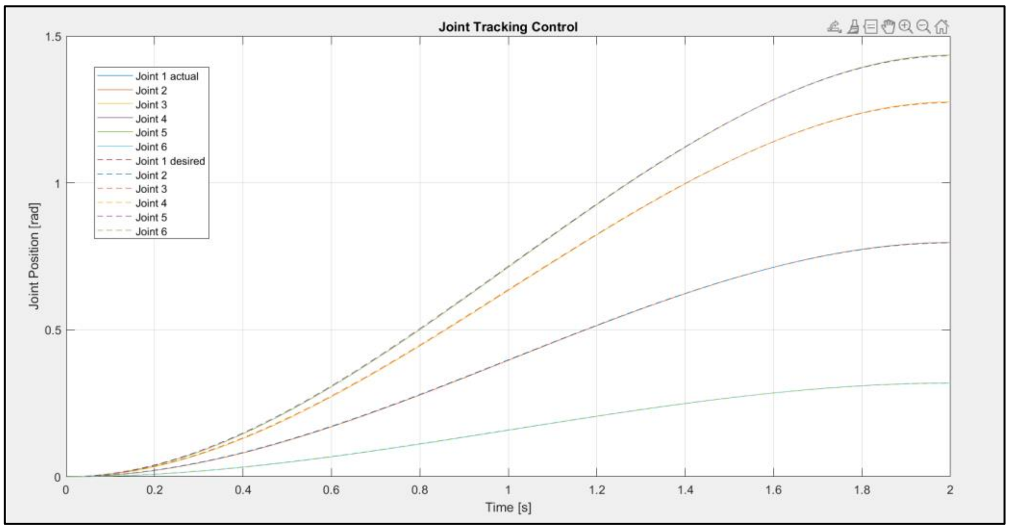


Figure 2. Joint position tracking

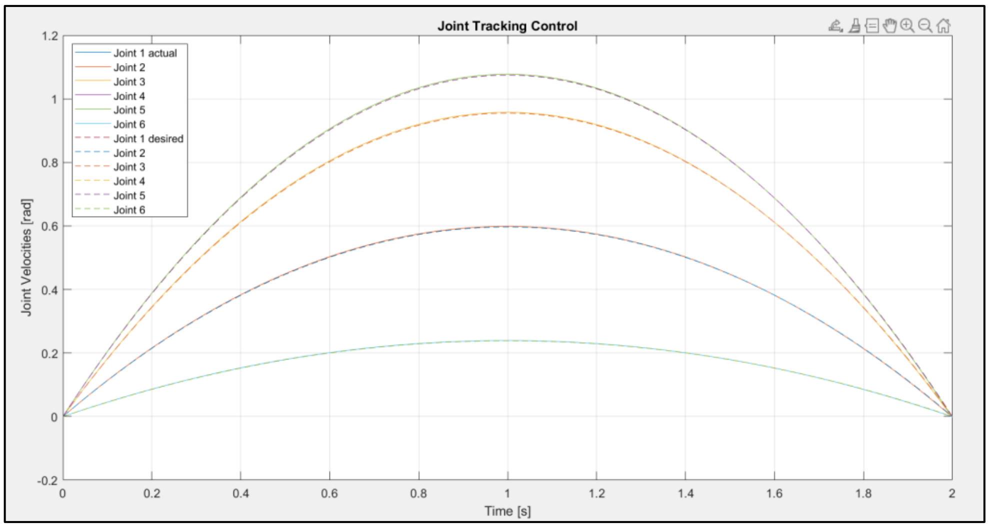


Figure 3. Joint velocity tracking

For what it concerns the second task, the robot must track a trajectory, from start to goal configuration, in an environment in which there are two spherical obstacles. The vectors defining the joint variables for the start and goal configurations are specified below, again starting and final velocities must be zero.

$$q(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \dot{q}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad q(3) = \begin{bmatrix} c_1/2\pi \\ c_1/2\pi \\ c_3/2\pi \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \dot{q}(3) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The core idea is to use a motion planning algorithm, specifically a Rapidly-exploring Random Tree (RRT), to compute a collision-free path between the two configurations. The path is then smoothed using appropriate post-processing techniques to improve continuity. Finally, a trapezoidal velocity profile is applied to the path to generate a time-parameterized trajectory, characterized by a constant velocity in the intermediate phase. The trapezoidal trajectory is obtained via the "*trapveltraj*" function. Below are some snippets from the MATLAB code developed for the second task.

```
tpts = 0:3;        %Time to track the trajectory
sampleRate = 20;   %Frequency of the controller
tvec = tpts(1):1/sampleRate:tpts(end);
numSamples = length(tvec);

planner = plannerBiRRT(ss,sv);

[path, solInfo] = plan(planner,startConfiguration,goalConfiguration);
interpolate(path, 250);

%Smoothing the path
path_smooth = cornercut(path, sv);
interpolate(path_smooth,250);

%Generate Trapezoidal Trajectory
[q, dq] = trapveltraj(path_smooth.States', numSamples);

%Visualize Trajectory
q = q';
disp(size(q));  % N Samples X DOF
disp(size(path_smooth.States))  % [N, 6]
rc = rateControl(sampleRate);
```

To verify whether the robot is in a valid configuration—i.e., not in collision and within its workspace—a manipulator state space and a state validator are defined. The state validator is configured with the option skippedSelfCollision = parent, allowing the robot to disregard self-collisions during validation. The validator is also passed to the motion planning algorithm to ensure that the generated paths are feasible and collision free. Additionally, a simple validation loop is implemented to check the feasibility of the goal configuration by verifying that it is collision-free and lies within the robot's reachable workspace. In the plot below there are reported the graphical results showing the joint positions and velocities over time for each joint.

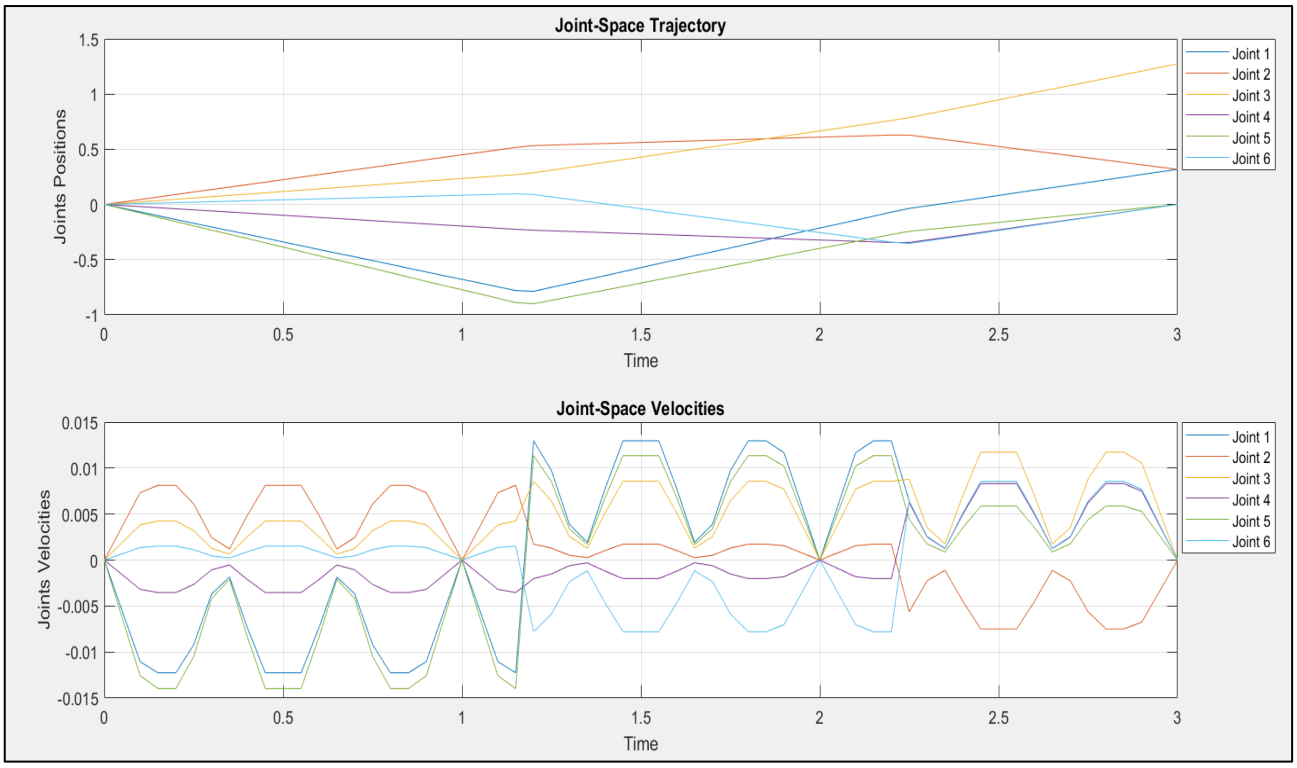


Figure 4. Trapezoidal trajectory

As shown, the joint positions converge to the desired final configuration, while both the initial and final velocities are zero, as required. The velocity profiles are approximately trapezoidal, confirming the expected three-phase structure of acceleration, constant velocity, and deceleration. The total duration of the trajectory is 3 seconds, as requested. As already shown for the previous task, a simple control strategy (PD and feedforward) should be implemented to actually control the robot to track the trajectory. For the sake of simplicity this step has been only carried out for the first task, but it is easily implementable following the same logic.

3. Differential drive tracking control

The aim of the second part of the project is to design and develop a controller in order to make the mobile robot track a desired circular trajectory, starting from an initial configuration whose variables define the position and the orientation of the robot in the plane. Below are some input data and the trajectory to be tracked:

- $x_{des}(t) = c_1 * \cos(c_2 t)$ where $c_1 = 2$ is the radius of the circular trajectory and $c_2 = 5$
- $y_{des}(t) = c_1 * \sin(c_2 t)$
- Distance between the wheels: $d = 0.4\ m$
- Radius of the wheels: $r = 0.2\ m$
- Start configuration: $q_i = (x_i, y_i, \theta_i) = (0, 0, 0)$

Furthermore is requested to compute the angular velocities of the wheels (ω_L, ω_R) and plot the cartesian position of the robot during the tracking. The idea is to implement Input/Output State Feedback Linearization (IO-SFL), defining a virtual control point (x_B, y_B) along the orthogonal axis to the one that joints the wheels. This point B is not subjected to any constraint (as the non-holonomic constraint of the mid-point) and, being rigidly connected to the robot at a distance b, a motion of B will determine a motion of the robot.

$$x_B = x + b * \cos(\theta) \qquad y_B = y + b * \sin(\theta)$$

Two virtual inputs are needed in order to simulate an actuation on the new control point. However, since they are not physically implementable, it is possible to compute, through the inverse of $T(\theta(t))$ matrix, the actual control input that has to be provided to the wheels of the robot in order to make the point B track the desired trajectory.

$$\dot{x}_B = v * \cos(\theta) - b\omega * \sin(\theta) = V_{dx} \qquad \dot{y}_B = v * \sin(\theta) + b\omega * \cos(\theta) = V_{dy}$$

$$\Downarrow$$

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = T(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = T^{-1}(\theta) \begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix}$$

Choose a point too far away from the robot could worsen the tracking, choose a too small value of b instead can lead to the saturation of the actuation system. Given a desired trajectory to be tracked, the virtual inputs are computed in order to impose an asymptotically stable error's dynamics, in such a way that after a transient the tracking error will tend to zero.

$$\begin{aligned} v_{dx} &= \dot{x}_{des} + k_1(x_{des} - x_B) \qquad & e_x &= x_{des} - x \qquad & \dot{e}_x + k_1 e_x &= 0 \qquad & e_x &\to 0 \\ v_{dy} &= \dot{y}_{des} + k_2(y_{des} - y_B) \qquad & e_y &= y_{des} - y \qquad & \dot{e}_y + k_2 e_y &= 0 \qquad & e_y &\to 0 \end{aligned}$$

The derivative of the trajectory is computed in order to implement feed-forward, common practice in trajectory tracking controller that allows obtaining much better performance. After couple trials, the control parameters have been set as $b = 0.8\ m$ (distance from mid-point) and proportional gain $k_1 =$

$k_2 = 10$ in order to have a faster convergence and a better tracking. After a short transient, the trajectory is successfully tracked. It is possible to show the Cartesian position of the robot during the tracking, saving at each simulation step the position of the robot. As expected, it follows a circular trajectory of radius $c_1 = 2$.

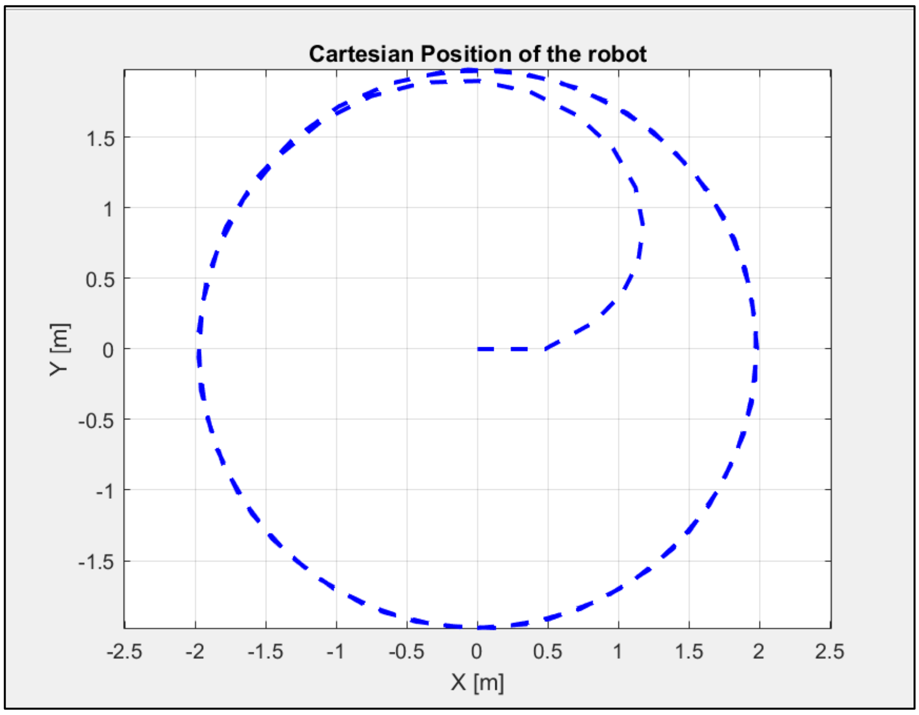


Figure 5. Cartesian position of the controlled robot

It is also possible to compute the angular velocity of each wheel, knowing the distance of the wheels and the radius. From the plot below it is possible to see that, after a short transient, the wheels reach a constant angular velocity that they keep during the tracking of the trajectory.

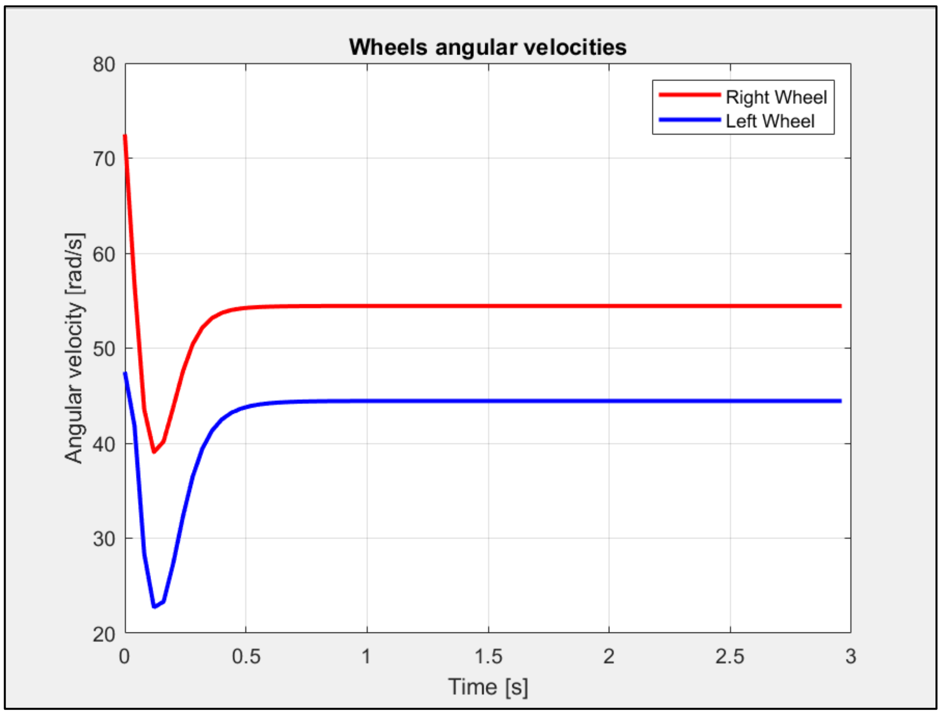


Figure 6. Angular velocities of the wheels

4. RobotStudio – Pick and Place in SSM

The purpose of this project is to simulate a robotic pick-and-place operation within a collaborative workspace using ABB's RobotStudio environment. The objective is to rearrange six cubic boxes, each measuring 4×4×4 cm, from a linear arrangement on a primary table to a 3×2 grid on a secondary table. To integrate both autonomous manipulation and human-robot collaboration, the task is executed implementing Speed and Separation Monitoring (SSM). Specifically, the robot reduces its speed to 0.2 m/s or stop when volumetric sensors detect a human, or more in general an object, in properly defined region of the workspace in the vicinity of the robot. The target is to achieve a complete pick-and-place cycle within 60 seconds under ideal, uninterrupted conditions. The workspace setup includes two tables with surfaces positioned 50 cm above the ground, separated by a physical barrier that extends 50 cm above the tabletop surface. A suitable industrial robot is selected from the RobotStudio library to ensure sufficient reach and functionality for the given task and is equipped with a smart gripper component. The very first step after the scene creation is to provide each object involved in the task with a properly oriented reference frame (work object). On each of the cubes to be moved the respective work object is oriented accordingly to the one on the robot's gripper. Once the setup phase is completed, the implementation of the pick and place routine is carried out.

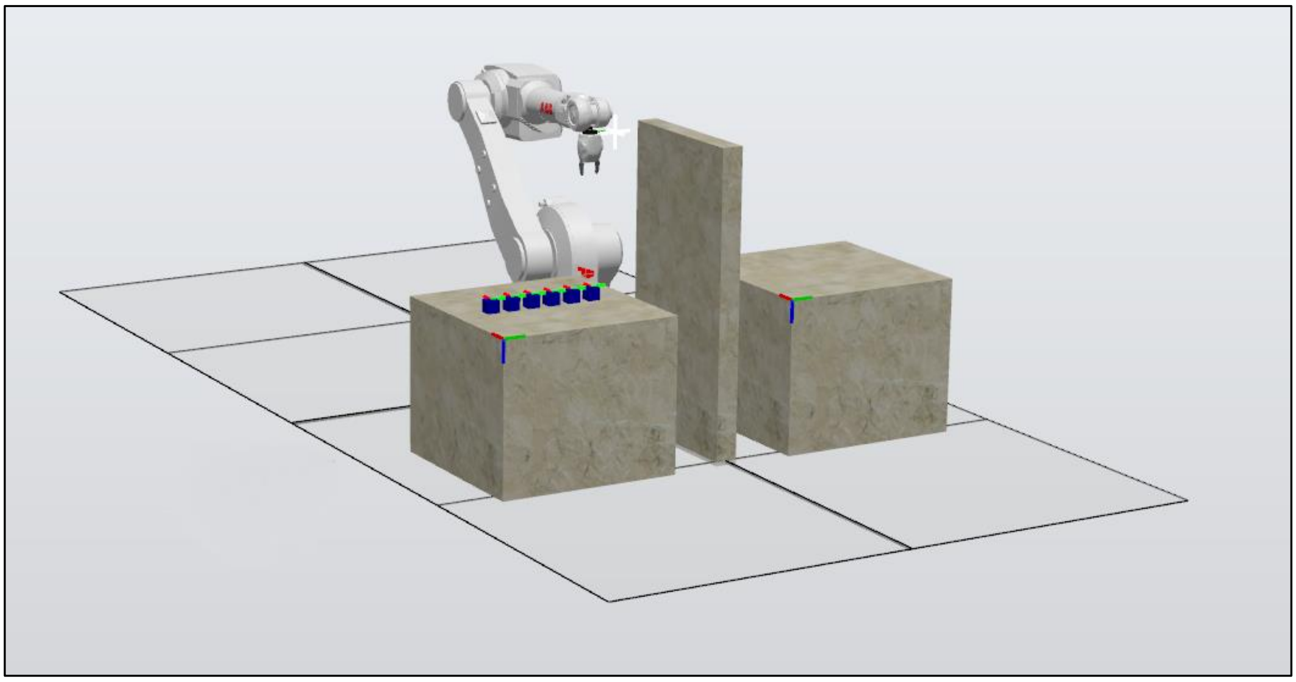


Figure 7. Setup in RobotStudio environment

4.1 Pick and Place routine

The task is structured around a sequence of pre-defined target configurations that serve as waypoints for the execution of the pick-and-place cycle. These targets are grouped based on their function within the routine and are taught to the robot in the RobotStudio environment in order to ensure repeatable motion and to eventually build the path to be followed. The target configurations are categorized into three main groups:

- **Pick** and **Pre-pick** targets: These targets are defined relative to the picking table. Each pick target corresponds to a position in which the robot gripper is precisely aligned with a box, ready to perform the grasping action. The associated pre-pick target lies directly above the corresponding pick point along the vertical axis (Z-axis). This configuration serves as a transitional waypoint, ensure a safe and collision-free descent towards the final picking configuration.
- **Home** and **Intermediate** targets (relative to WObj0): The home position represents the robot's default configuration at the beginning and end of a cycle. It provides a safe and neutral pose for initialization and closing procedures. The intermediate target is a strategically placed waypoint that assists the robot in bypass the central barrier separating the two tables. It is also used to facilitate the homing procedure.
- **Place** and **Pre-place** targets: Similar to the picking phase, the placing operation includes place targets where the gripper aligns with the target position on the secondary table to release the box. The corresponding pre-place targets are vertically offset positions, again to facilitate the approach to the placing position.

Once the target configurations are defined, the pick-and-place routine is structured and implemented as a sequential series of movements executed for each individual box. The motion sequence follows this order:

1. Move to the intermediate target
2. Move to the pre-pick target
3. Move to the pick target (grasping action occurs)
4. Retract to pre-pick
5. Move to intermediate
6. Move to pre-place
7. Move to the place target (release action occurs)
8. Retract to pre-place
9. Return to intermediate

All movements between targets are set as joint movements with the interpolation setting set to FINE. This option ensures that the robot reaches each target position with high precision before proceeding to the next. The velocity of each motion is set in the RAPID Module, in order to be eventually able to dynamically changing it. In addition, a homing procedure is implemented and it is executed before initiating the pick-and-place cycle. This ensures that the robot begins each cycle from a known, safe starting configuration. To facilitate the reset of the environment after completing a full cycle, an inverse pick-and-place routine is created. This secondary sequence uses the same target positions in reverse order to return the boxes from the placing table back to their original positions on the picking table. The resulting scene is shown in the figure below, each target configuration is oriented accordingly to the gripper orientation. The robot is in the home configuration, it is possible to easily noticing the intermediate target and all the other ones involved in the path.

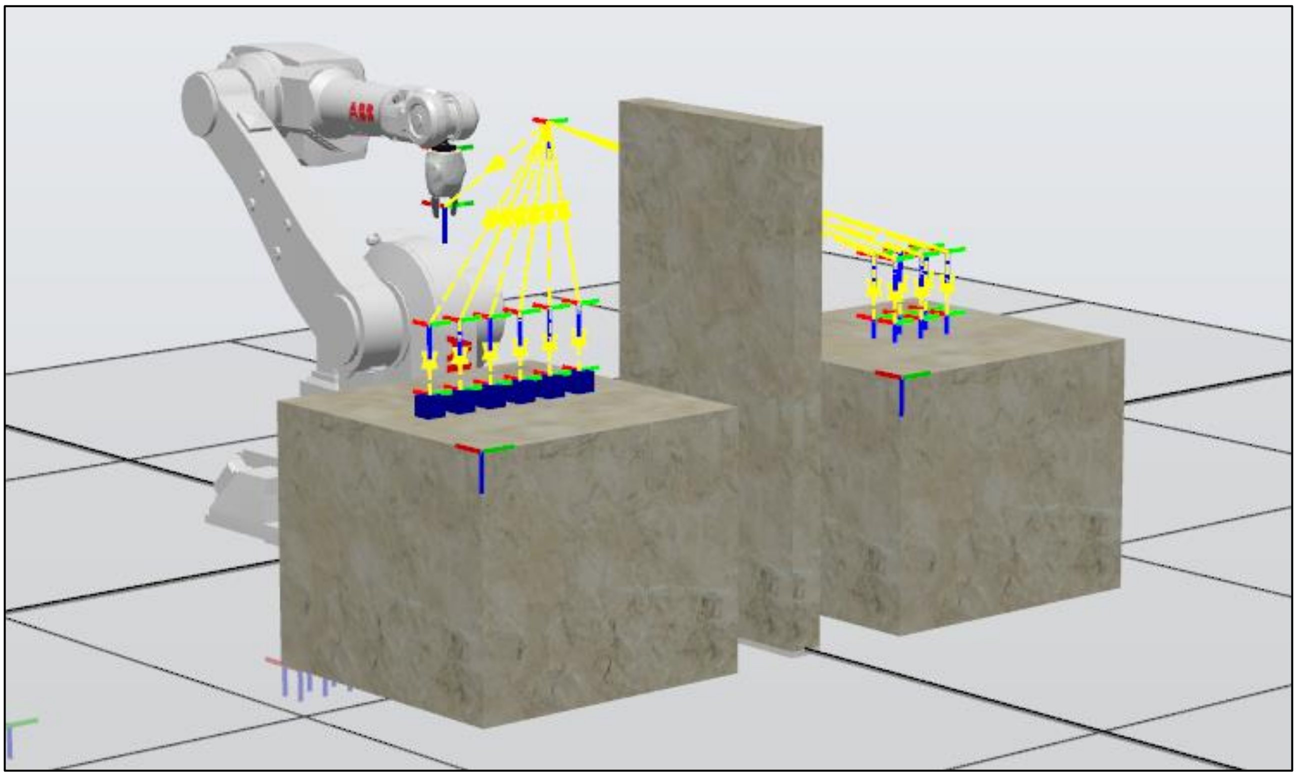


Figure 8. Targets and path

4.2 Station logic, I/O Signals and sensors

To enable the robot to grasp, transport, and release the boxes during the pick-and-place routine, a dedicated Smart Component was created to simulate the gripper's behavior. This component is connected to the robot controller through appropriately configured Digital I/O signals, allowing it to be triggered programmatically during the routine execution. Inside the gripper Smart Component, several key elements are implemented:

- A Line Sensor, positioned near one of the gripper fingers, detects the presence of a box when it comes into contact.
- Two components, an Attacher and a Detacher, are used to simulate the physical grasping and releasing of the box by respectively attaching it to and detaching it from the gripper.
- Two Joint Movers simulate the mechanical actuation of the gripper fingers—one responsible for closing the gripper and the other for opening it.
- A Logic SR Flip-Flop (Set/Reset memory element) to maintain the gripper in the current state until an explicit reset signal is given.
- A NOT operator that inverts the Digital Output of the controller provided to the gripper component and it sends it to the detacher.

The logic is as follows: when the controller sends a high signal (1) via the designated Digital Output, the gripper initiates a closing motion through the joint mover until it reaches a predefined position. If, during this motion, the Line Sensor detects the box, the Attacher is immediately triggered, fixing the box to the gripper and enabling its transportation. When the controller send a low signal (0) to the gripper component, it is inverted through a NOT operator, triggering the detaching of the box and, consequentially, the gripper's opening. Following this logic, it is sufficient to set, in the RAPID module, the digital output (DO_Gripper) to its high value (1) as soon as the gripper reaches the pick configuration. The output signal of the gripper is fed back to the input of the controller, so as soon as DI_Gripper is equal to (1), the box is attached and the robot can move to the next target. Following the same reasoning, as soon as the robot reaches the place target, the digital output (DO_Gripper) is set to zero, in order to activate the detaching of the box followed by the opening movement. The box is therefore placed in the target position and the robot can move towards the next target configuration.

```
PROC Path_10()

    MoveJ intermediate,speed,fine,Servo\WObj:=wobj0;
    MoveJ pre_pickup_1,speed,fine,Servo\WObj:=table1_wobj;
    MoveJ pickup_1,speed,fine,Servo\WObj:=box1_wobj;
    WaitTime 1;
    SetDO DO_Gripper,1;
    WaitDI DI_Gripper,1;
    MoveJ pre_pickup_1,speed,fine,Servo\WObj:=table1_wobj;
    MoveJ intermediate,speed,fine,Servo\WObj:=wobj0;
    MoveJ pre_place_1,speed,fine,Servo\WObj:=table2_wobj;
    MoveJ place_1,speed,fine,Servo\WObj:=goal_1;
    WaitTime 1;
    SetDO DO_Gripper,0;
    WaitDI DI_Gripper,0;
```

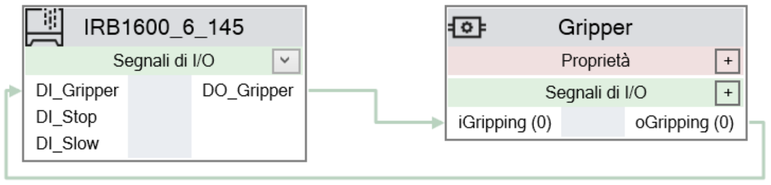


Figure 9. Controller and Gripper logic

All the necessary digital input/output signals are built in the controller configuration. The digital inputs DI_Stop and DI_Slow are defined to implement Speed and Separation Monitoring mode of operation.

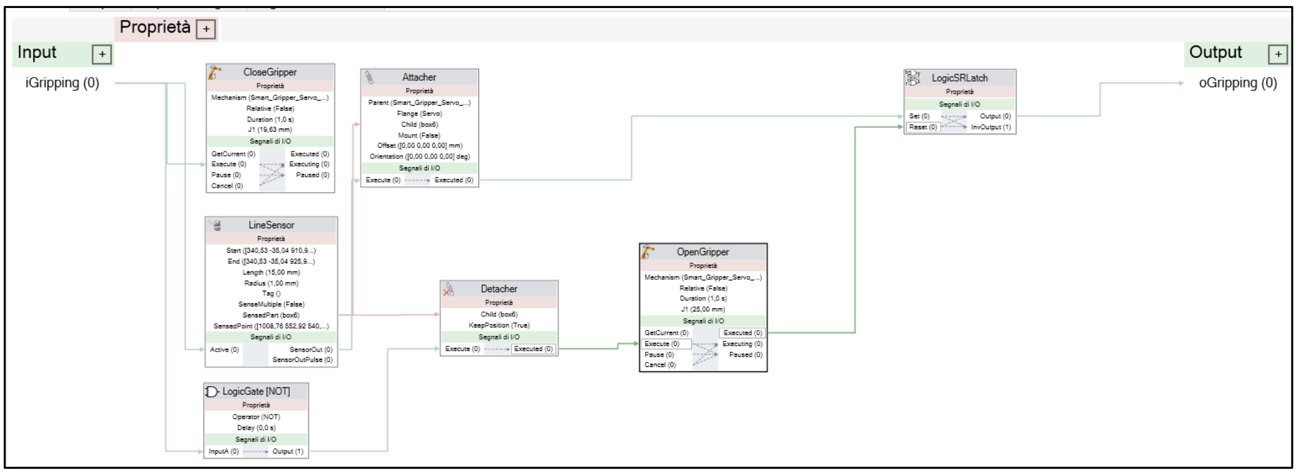


Figure 9. Gripper's inner logic

4.3 Speed and separation monitoring (SSM)

To ensure safe human-robot collaboration, the project implements Speed and Separation Monitoring by defining sensorized zones that dynamically adjust the robot's behavior when a human or object is detected within the workspace. The objective is to reduce the robot's speed or even stop it, when a human or, more in general, an object is sensed by the sensors. The first step involves creating a Smart Component representing the "red zone", which corresponds to the high-risk area immediately surrounding the robot. When a presence is detected within this volume, the robot must stop all operations for safety. This component includes:

- A Volumetric sensor that detects the intrusion of an object or person within its defined space.
- A NOT Gate, used to invert the sensor signal logic so that the robot stops only when the zone is occupied.
- A Logic SR Flip-Flop (Set/Reset memory element) to latch the stop condition, maintaining the robot in a halted state until an explicit reset signal is given.

The logic is wired to the controller through a digital input (DI_Stop). When this input is set to 1 (indicating that a presence has been detected in the red zone) a TRAP routine named "RedZone" is triggered within the RAPID programming environment. This TRAP routine is designed to immediately stop the robot's motion and keep it in a stopped state until the signal is reset. Once the digital input returns to 0 (meaning the red zone is clear), the routine allows the robot to resume its task from where it left off. Following a similar approach, a yellow zone is implemented to allow the robot to continue its operation at a reduced speed (specifically, 0.2 m/s) when a presence is detected within a predefined area surrounding the red zone. A dedicated smart component is created to represent this yellow safety zone. This component is connected to the robot controller via appropriate Digital Input (DI_Slow), and it integrates the following elements:

- Volumetric sensors, exactly as for the red area.
- OR operators, used to combine the outputs of multiple sensors, allowing the zone to trigger a response even if only one area is occupied.
- A NOT operator, used to generate the complementary signal needed for reset conditions;
- A LogicSR block, which stores the state and ensures the signal holds until a reset is performed.

When the signal from this zone is activated, it informs the controller via the DI_Slow signal. The robot then automatically reduces its speed, while continuing its task in a controlled manner. Another trap routine named "ExitYellowZone" is triggered through a recover signal, as soon as the yellow area is uncluttered again. The final logic of the station is shown below.

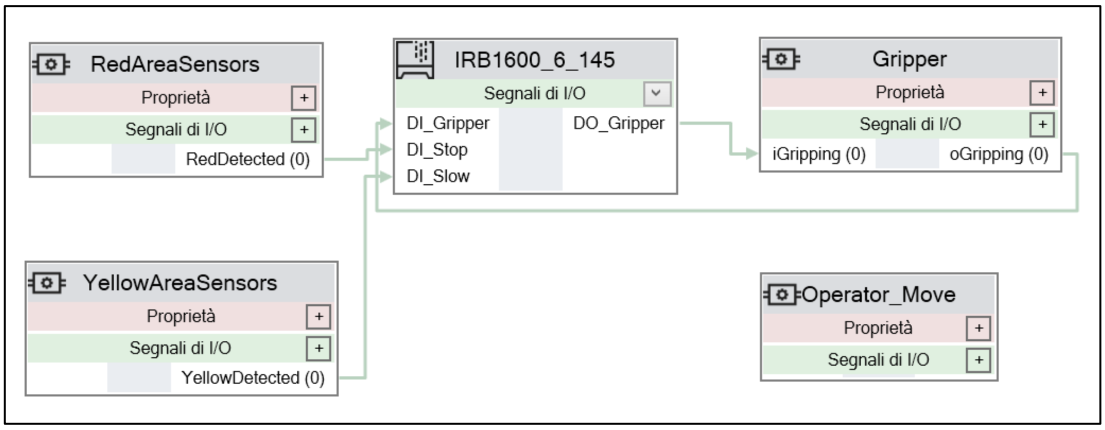


Figure 11. Final logic of the station

The final implementation is shown below. The volumetric sensors are hided and embedded in the red and yellow areas for visualization purpose. A cylindrical body is moved along a curve to simulate a real scenario in which an operator approaches the robot and enters the safety areas, in order to test the actual operation of the SSM mode of operation implemented so far.

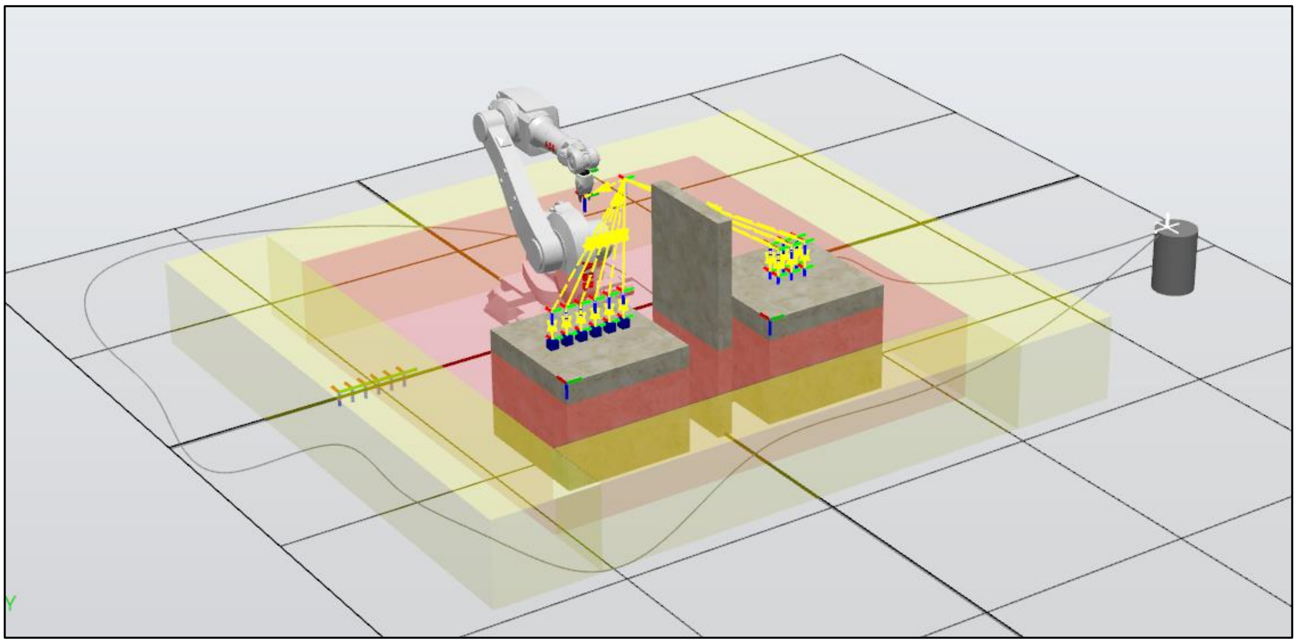


Figure 12. Final set - Pick and Place in SSM

From visual simulation, it is possible to assess the proper operation of the robot, according to the project specifics. The pick and place task, in ideal condition, namely without the intrusion of the operator in none of the safety zones, it is fully completed in much less than 60 seconds, with the robot moving at 2 m/s. The Speed and Separation Monitoring is correctly implemented, the robot adjust its speed to 0.2 m/s as the yellow area sensors are triggered and it completely stops when the operator is detected in the red area.