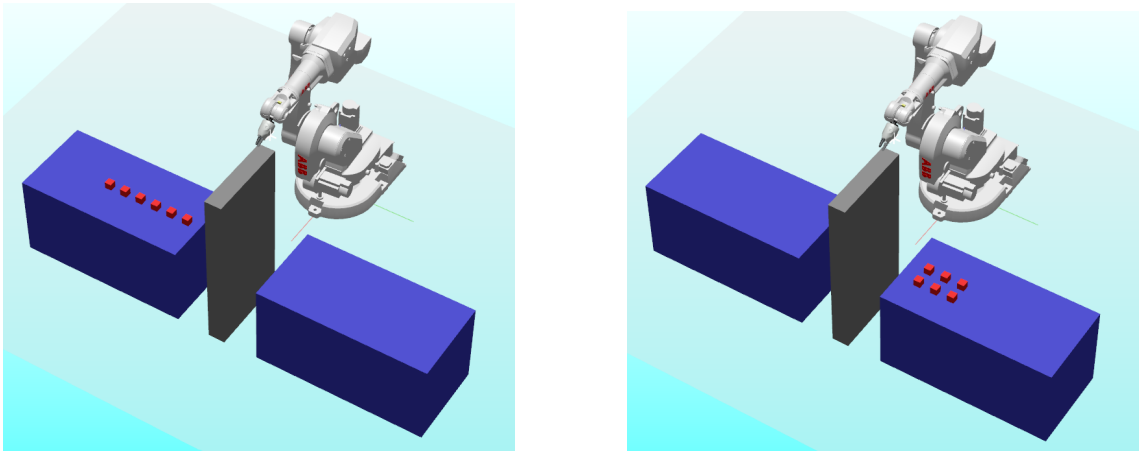
3 Project 3: ABB Robot Studio

The goal of this project is to develop a robotic system capable of rearranging six 4x4x4 cm boxes from a linear formation on a primary table into a 3x2 grid layout on a secondary table as illustrated in the figure below.



(a) Initial Situation (b) Final Situation

Figure 17: Goal of the Project

The project will be divided into two main phases:

→ **Phase 1 – Development of the basic pick and place operation:** In this initial phase, a suitable robot will be selected from the RobotStudio catalog, considering the required reach. The basic motion logic for transferring the boxes from the primary table to the secondary table will be implemented.

→ **Phase 2 – Integration of Speed and Separation Monitoring:** In the second phase, collaborative functionality will be introduced by enabling *Speed and Separation Monitoring* mode. Within the shared workspace, the robot's speed will be limited to 0.2 m/s to ensure human safety.

The physical setup for the simulation includes the following:

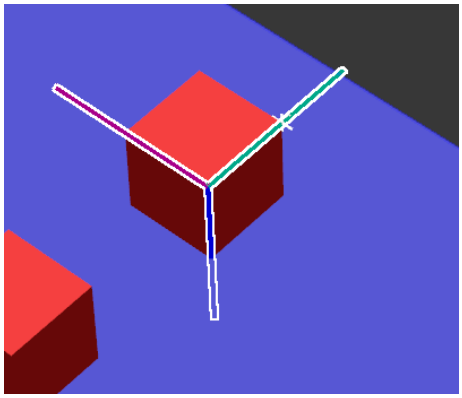
- **Table height**: Both the primary and secondary tables will have their surfaces set at a height of 50 cm from the ground.
- **Separation**: A vertical barrier, 100 cm in height, will be placed between the two tables.
- **Cycle time**: The target cycle time for completing the full operation without disturbances is 60 seconds.

3.1 Pick and Place

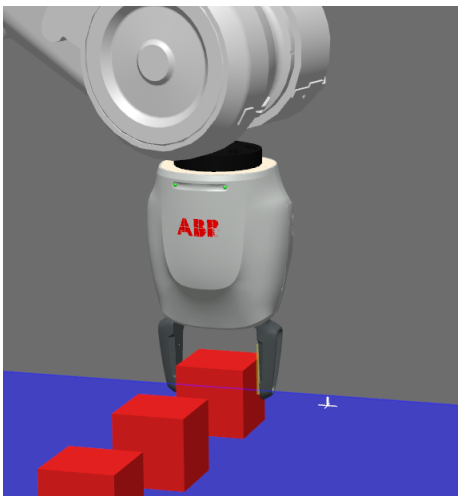
In this initial phase, the simulation environment was created by positioning the two tables, the separating barrier, and the six boxes to be moved. A suitable robot path was generated, and the control logic was implemented to allow the gripper to pick up each box from the primary table and place it onto the secondary table according to the target configuration.

3.1.1 Environment Setup

The first step consisted in building the simulation environment, which includes the two tables, a vertical barrier separating them, and six boxes arranged in a linear formation.
For the manipulator, the IRB 1600 model was selected, featuring a maximum payload of 6 kg and a reach of 145 cm—characteristics that fulfill the operational requirements of the task. Subsequently, reference frames were defined for both tables and for each individual box to ensure precise positioning and manipulation within the simulation.
For each box, a reference frame was defined with the Z-axis oriented downward, allowing the gripper to grasp the boxes from the top surface.



(a) Box's Reference Frame



(b) Grip the box by its upper side

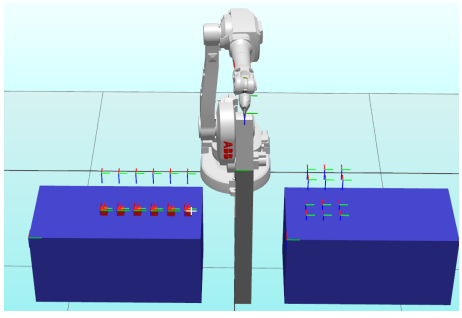
Figure 18: Choice of reference frame for the boxes

3.1.2 Generate the Path

To advance to the subsequent stage of path generation, it is essential to define the Path Targets. For each box i , the following targets have been established:

- **Pick_Box_i**: The target's origin is positioned at the centroid of the upper surface of Box i and is aligned with the orientation of Box i's reference frame.
- **Pre_Pick_Box_i**: This target is oriented identically to the **Pick_Box_i** target and is translated 30 cm above Box i.
- **Place_Box_i**: This target is oriented identically to **Pick_Box_i** target and is translated along the x and y axes to achieve the desired position on Table 2 of Box i.
- **Pre_Place_Box_i**: This target shares the same orientation as the **Place_Box_i** target and is translated 30 cm vertically above it.

Additionally, an **Upper_Barrier** target has been defined above the barrier to ensure collision avoidance during the pick-and-place operation.
Finally, a **Homing** target has been specified to enable the manipulator's homing procedure.



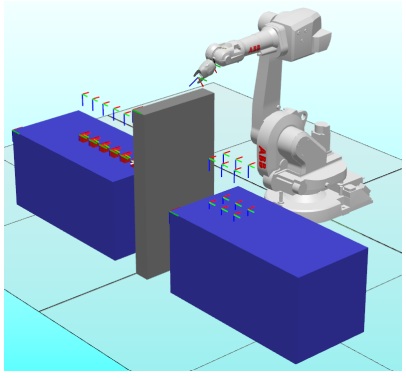


Figure 19: Target of the Path

After creating the targets, it is possible to generate the path. The general sequence of the path follows the logic:

$$\ldots \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J} \textbf{Pre\_Pick\_Box\_i} \xrightarrow{L} \textbf{Pick\_Box\_i} \xrightarrow{L} \textbf{Pre\_Pick\_Box\_i} \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J}$$
$$\xrightarrow{J} \textbf{Pre\_Place\_Box\_i} \xrightarrow{L} \textbf{Place\_Box\_i} \xrightarrow{L} \textbf{Pre\_Place\_Box\_} \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J} \ldots$$

Where **L** denotes a linear movement in Cartesian space and **J** denotes a joint movement, as defined in the path programming conventions of RobotStudio.

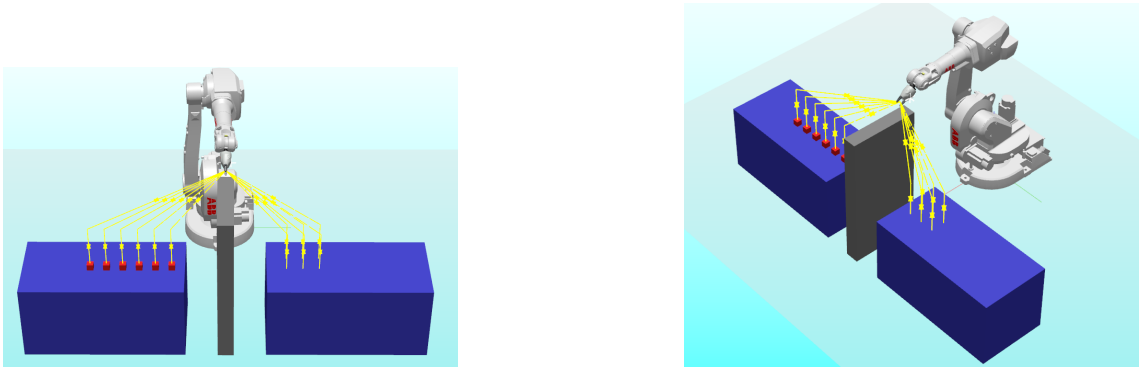


Figure 20: Generated Path

Appendix A presents the RAPID code corresponding to the path generated for executing the pick-and-place operation of the generic Box 2.

3.1.3 Design and Integration of a Sensorized Gripper

To successfully complete the pick-and-place task, it is necessary to integrate a sensor on the gripper, enabling the system to detect and grasp the box from Table 1 and accurately place it onto Table 2.

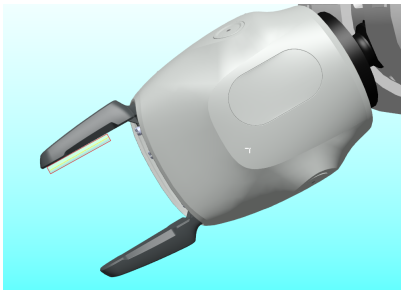
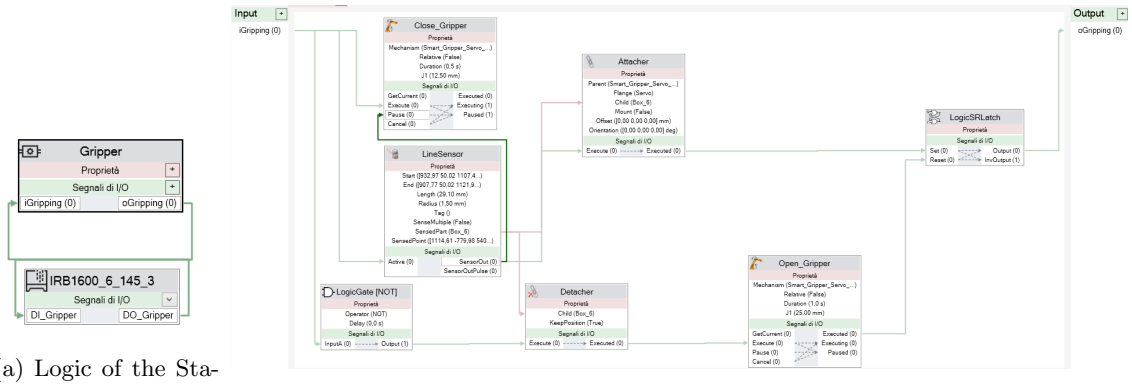


Figure 21: Sensor on Finger of Gripper

The logic governing the gripper operation is presented in the following figure.



(a) Logic of the Station

(b) Logic of the Gripper

Figure 22: Logic of System

In image (b), the highlighted arrow has been added with respect to the exercise carried out during the course to prevent the gripper from crushing the box.

Appendix B presents the RAPID code corresponding to the pick-and-place phase for the generic Box 2.

3.2 Speed and Separation Monitoring

Before implementing speed and separation monitoring, it is necessary to implement reverse pick-and-place operation to allow a simulation that does not terminate once the six boxes have been placed on Table 2.
The logic is the same as described in Section 3.1; the only difference is the path, which for each box will start from Table 2 and end at Table 2.
To avoid creating duplicate targets, the following target reuse strategy is adopted:

- The **Pre_Place_Box_i** target is used as the pre-pick position on Table 2;
- The **Place_Box_i** target is used as the pick position on Table 2;
- The **Pre_Pick_Box_i** target is used as the pre-place position on Table 1;
- The **Pick_Box_i** target is used as the place position on Table 1.

In this case, the resulting general path sequence follows the logic below:

$$\ldots \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J} \textbf{Pre\_Place\_Box\_i} \xrightarrow{L} \textbf{Place\_Box\_i} \xrightarrow{L} \textbf{Pre\_Place\_Box\_i} \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J}$$
$$\xrightarrow{J} \textbf{Pre\_Pick\_Box\_i} \xrightarrow{L} \textbf{Pick\_Box\_i} \xrightarrow{L} \textbf{Pre\_Pick\_Box\_} \xrightarrow{J} \textbf{Upper\_Barrier} \xrightarrow{J} \ldots$$

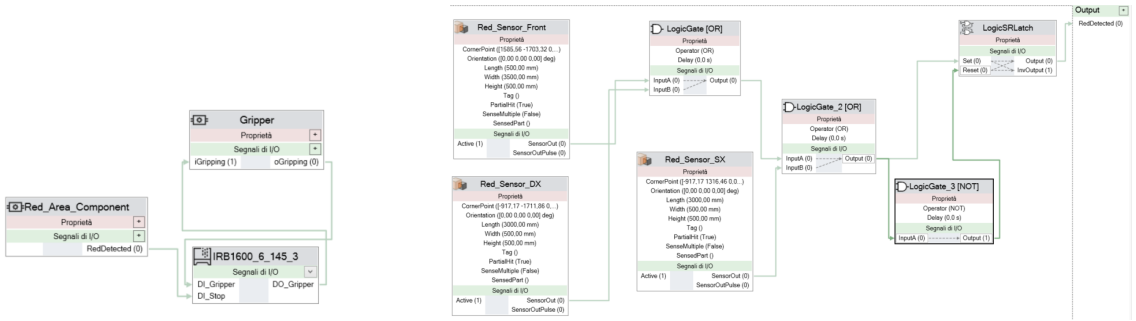
Appendix C presents the RAPID code corresponding to the reverse pick-and-place phase for the generic Box 2.

3.2.1 Red Zone

The red zone was the first to be defined. When the operator enters this area, the manipulator is immediately stopped.
It is assumed that the robot cannot be reached by the operator from the rear side, due to the presence of a fence at the back, as shown in the following figures.

Figure 23: Red Zone

The control logic responsible for Red Zone detection is presented in the following figure.



(a) Logic of the Station (b) Logic of the Red Zone Detection

Figure 24: Logic of System with Red Zone

Appendix D provides the RAPID code associated with the implementation of the Red Zone.

3.2.2 Yellow Zone

The yellow zone was the last to be defined. When the operator enters this area, the manipulator's speed is limited to 0.2 $\frac{m}{s}$.

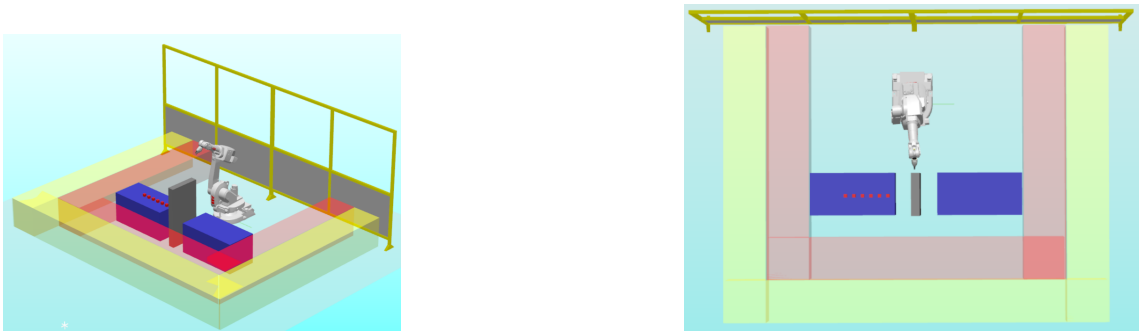
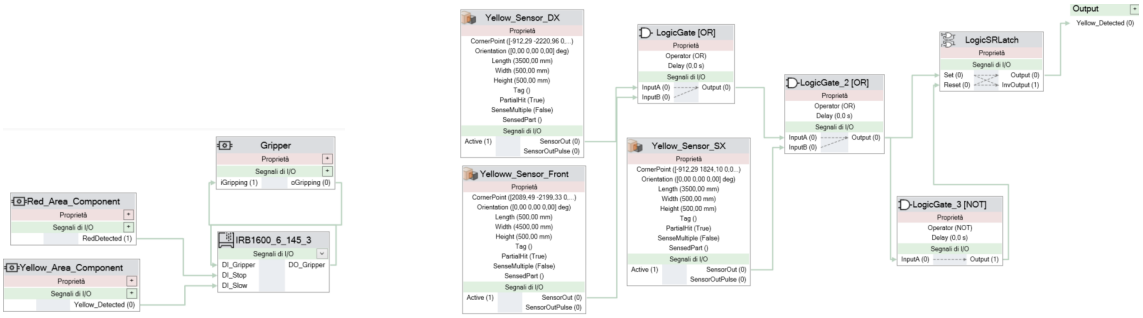


Figure 25: Red and Yellow Zone

The control logic responsible for Yellow Zone detection is presented in the following figure.



(a) Logic of the Station (b) Logic of the Yellow Zone Detection

Figure 26: Logic of System with Red Zone and Yellow Zone

Appendix E provides the RAPID code associated with the implementation of the Red Zone and also the Yellow Zone.

3.2.3 Mobile Operator Implementation

To simulate a virtual operator moving near the robot, a CAD model sourced online was imported. The next step involved creating the path that the operator must follow during movement; this path was generated using spline techniques and is depicted in blue in the following figure.

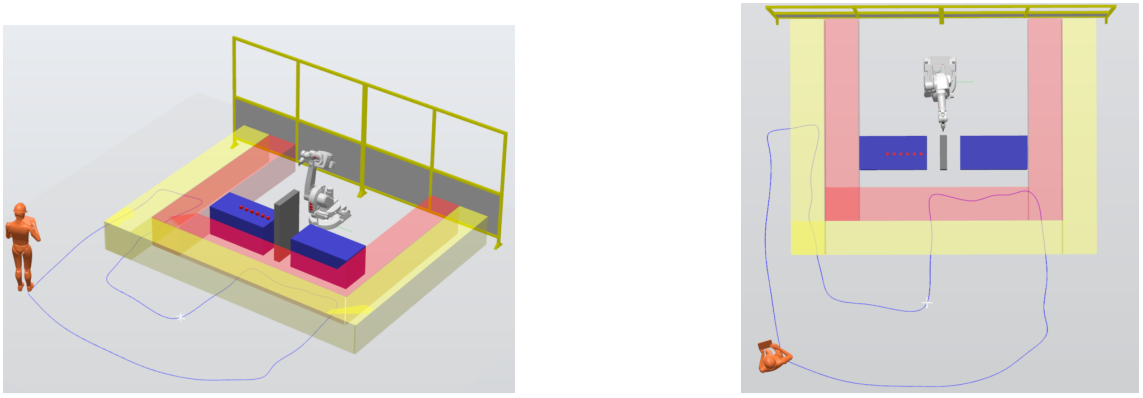


Figure 27: Red and Yellow Zone

To automate the operator's movement, the smart component **MoveAlongCurve** is employed with the following parameters:

- Imported Operator's CAD as the Object parameter,
- The spline-generated curve as the WirePart parameter,
- A speed of 0.3 $\frac{m}{s}$ for the operator.

As a final refinement, the Red Zone can be adjusted to cover the entire working area in the immediate vicinity of the robot, including the rear side. This ensures that the manipulator does not resume task execution if the operator has passed through both the Yellow Zone and the Red Zone, as illustrated in the following figure.

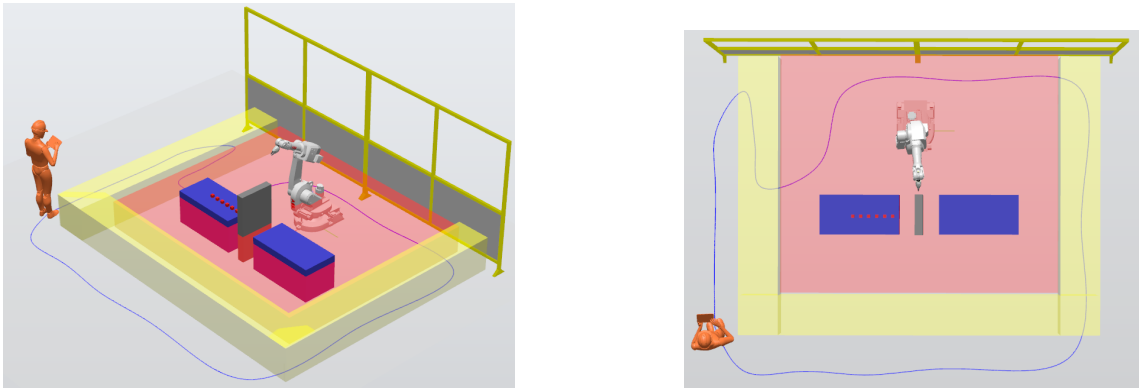


Figure 28: New Red Zone Implementation

As for the system logic, the only modification involves the logic of the Red_Zone component.

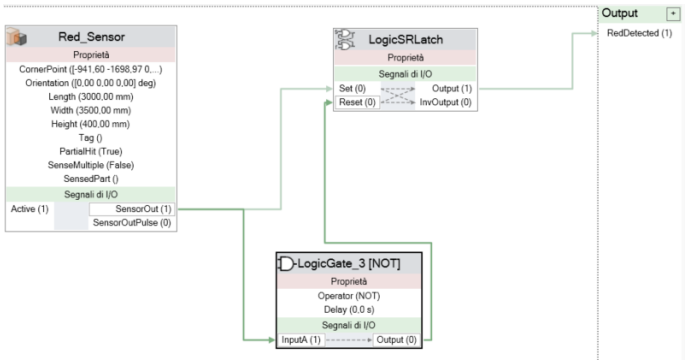


Figure 29: Logic of the new Red Zone Detection

4 Appendix

4.1 Appendix A

```
PROC PickAndPlace()
        ! Box 2
        MoveJ Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveL Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveL Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
        MoveJ Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveL Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveL Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
ENDPROC
```

Listing 1: RAPID code for path generation

4.2 Appendix B

```
PROC Pick_Place()
        ! Box 2
        MoveJ Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveL Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        WaitTime 1;
        SetDO DO_Gripper,1;
        WaitDI DI_Gripper,1;
        MoveL Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
        MoveJ Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveL Place_2_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        WaitTime 1;
        SetDO DO_Gripper,0;
        WaitDI DI_Gripper,0;
        MoveL Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
ENDPROC
```

Listing 2: RAPID code for pick-and-place operation

4.3 Appendix C

```
PROC Reverse_Pick_Place()
        ! Box 2
        MoveJ Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveL Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        WaitTime 1;
        SetDO DO_Gripper,1;
        WaitDI DI_Gripper,1;
        MoveL Pre_Pick_Box_2,speed,fine,Servo\WObj:=RF_Tab_1;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
        MoveJ Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveL Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        WaitTime 1;
        SetDO DO_Gripper,0;
        WaitDI DI_Gripper,0;
        MoveL Pre_Place_Box_2,speed,fine,Servo\WObj:=RF_Tab_2;
        MoveJ Upper_Barrier,speed,fine,Servo\WObj:=wobj0;
ENDPROC
```

Listing 3: RAPID code for reverse pick-and-place operation

4.4 Appendix D

```
VAR speeddata speed;
VAR intnum iStop;
PROC main()
    IDelete iStop;
    CONNECT iStop WITH RedZone;
    ISignalDI DI_Stop, 1, iStop;

    speed:= v2000;
    WHILE TRUE DO
        Homing;
        Pick_Place;
        Reverse_Pick_Place;
        Homing;
    ENDWHILE
ENDPROC

TRAP RedZone
    StopMove;
    WaitDI DI_Stop, 0;
    StartMove;
ENDTRAP
```

Listing 4: RAPID code for Red Zone Implementation

4.5 Appendix E

```
VAR speeddata speed;
VAR speeddata high_speed:= v2000;
VAR speeddata low_speed := v200;
VAR intnum iStop;
VAR intnum iSlow;
VAR intnum iRecover;

PROC main()
    IDelete iStop;
    CONNECT iStop WITH RedZone;
    ISignalDI DI_Stop, 1, iStop;

    IDelete iSLow;
    CONNECT iSLow WITH YellowZone;
    ISignalDI DI_Slow, 1, iSLow;

    IDelete iRecover;
    CONNECT iRecover WITH ExitYellowZone;
    ISignalDI DI_Slow, 0, iRecover;

    speed:= high_speed;
    WHILE TRUE DO
        Homing;
        Pick_Place;
        Reverse_Pick_Place;
        Homing;
    ENDWHILE
ENDPROC

TRAP RedZone
    StopMove;
    WaitDI DI_Stop, 0;
    StartMove;
ENDTRAP

TRAP YellowZone
    StopMove;
    speed:= low_speed;
    StartMove;
ENDTRAP
```

Listing 5: RAPID code for Red Zone and Yellow Zone Implementation