HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Industrial Robotics Lab Report

Group 2A

Student Names	Matriculation Number
Arjun Rameshbhai Beladiya	28744163
Dhaval Jagdish Mistry	20644015
Tarang Chimanbhai Italiya	12543942

July 4, 2025

Prof. Dr.-Ing. Konrad Wöllhaf

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Contents

List of Figures and Flowcharts

Lab Test 1 Teaching Tool and Base, Simple Program

1.1 Introduction

This lab focused on practical training with KUKA industrial robots, which are widely used in automation due to their precision, flexibility, and ability to perform complex tasks. The main goal was to gain hands-on experience in operating and programming these robots while following strict safety procedures. The tasks included moving the robot in different coordinate systems (axes, world, tool), teaching the tool center point (TCP), teaching the base coordinates of a box, and creating a robot program that moves the tool tip along the box's edges. Additionally, the lab explored how the robot program can adapt when the box's position is changed by re-teaching and updating the base coordinates.

1.2 Objective

- Move the robot in the diferent coordinate systems (axes, world, tool)
- Teach the tool
- Teach the base of the box
- Code the tool tip to trace the box edges
- Moves the box, then re-teaches and updates its base values.

1.3 Methodology

1.3.1 Teach the Tool using 4-point method

To teach the tool, the robot is jogged to four points in space where the tool tip will be located. The KUKA robot uses a 4-point method for this purpose.

1. Specify the tool number and name. Select tool number 5 and use a name of your choice. Confirm with Continue or Ok.

2. The dialog then appears with the request to align the tool by moving the robot to the first reference point. The goal is for the tip of the tool to point to the same place from four different directions.

3. Move the robot to the first reference point as shown in Figures 1.7 and 1.8, and press Continue or Ok.

4. Repeat this for the next three reference points, ensuring that the tool tip is aligned correctly each time.

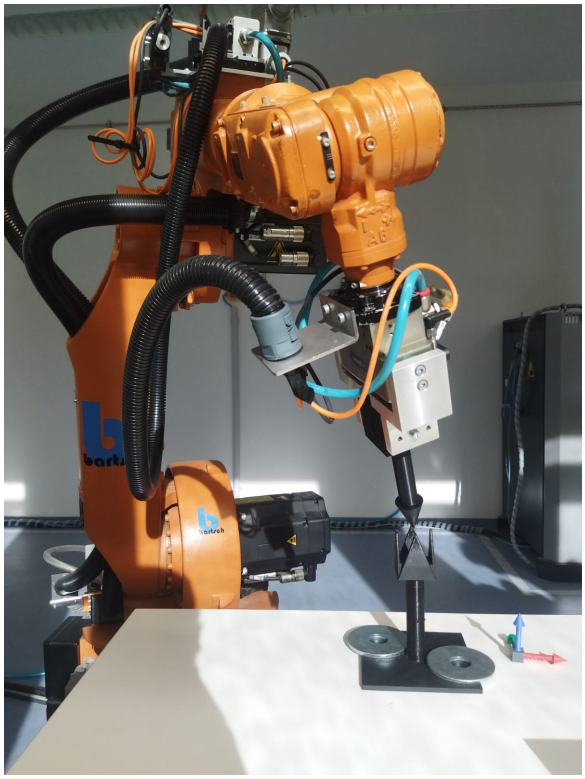


Figure 1.1: First reference point

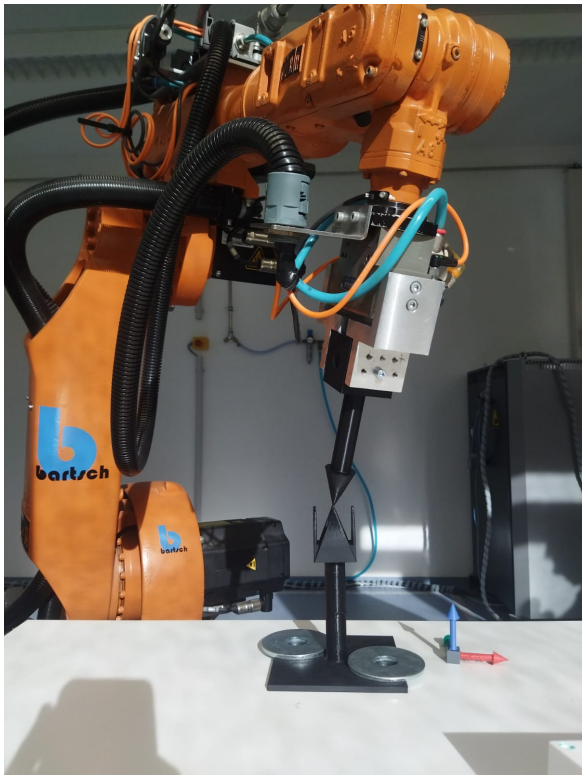


Figure 1.2: Second reference point

Figure 1.3: Fourth reference point

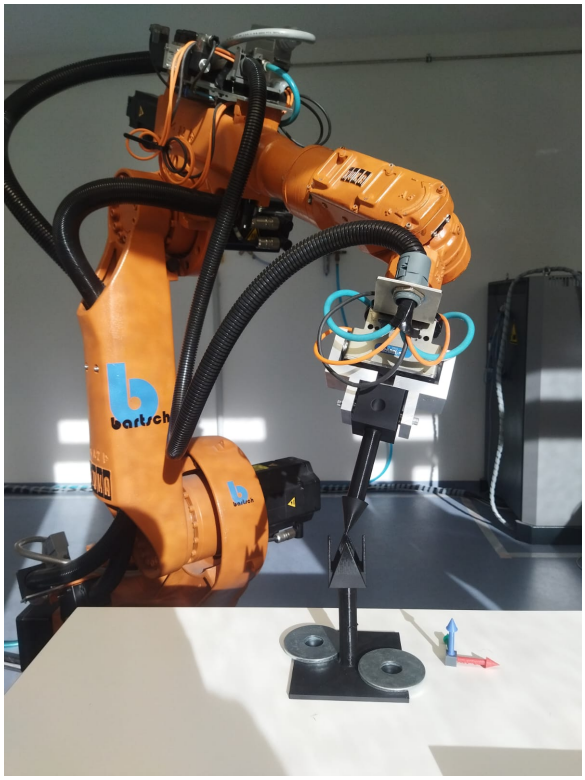


Figure 1.4: Third reference point

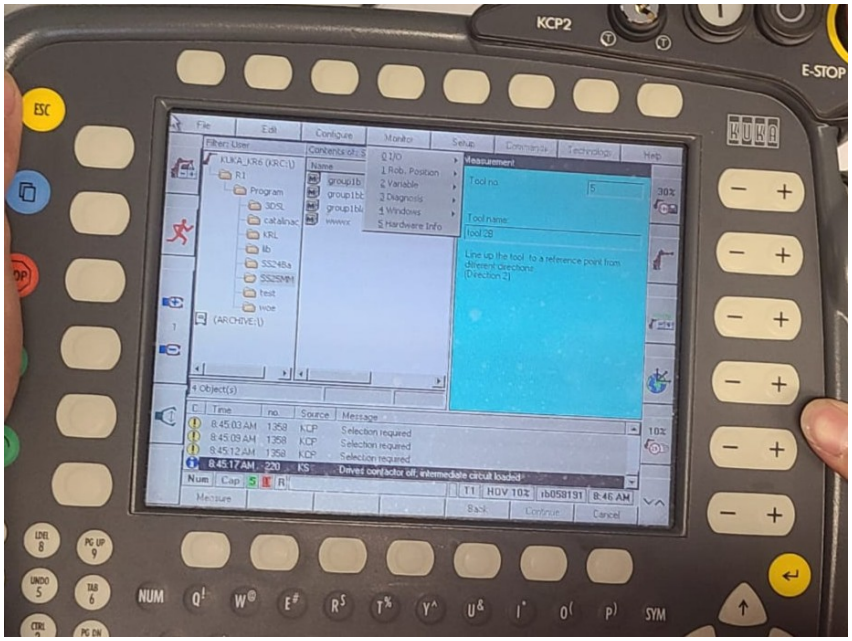


Figure 1.5: Tool teaching dialog

5. After the fourth point, the tool is considered taught. The robot will calculate the tool's position and orientation based on these points.

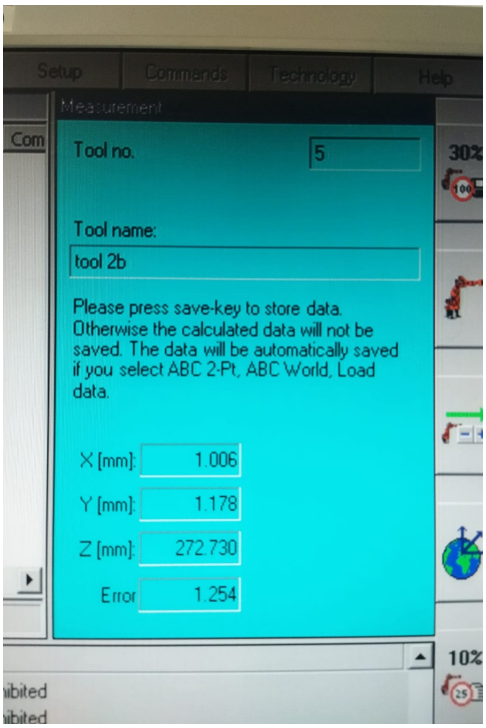


Figure 1.6: Tool Accuracy

1.4 Teaching the Base of the Box

1. The supervisor first positioned the box on the workbench.

2. Afterwards, we selected the tool that we had previously taught and used it to touch key corners of the box.

3. Next, we recorded the coordinates of three non-collinear points: the origin, a point along the X-axis, and a point in the XY-plane to define the Y-axis.

4. Using these reference points, we were able to define and teach the base coordinate system of the box successfully.

1.4.1 Base Calibration

1. Once the base was taught, we verified the calibration by moving the robot's tool tip to the recorded points and checking the alignment.

2. Finally, we fine-tuned the base coordinates as needed to ensure that the tool tip accurately followed the edges of the box.

Figure 1.7: Position of work piece on table

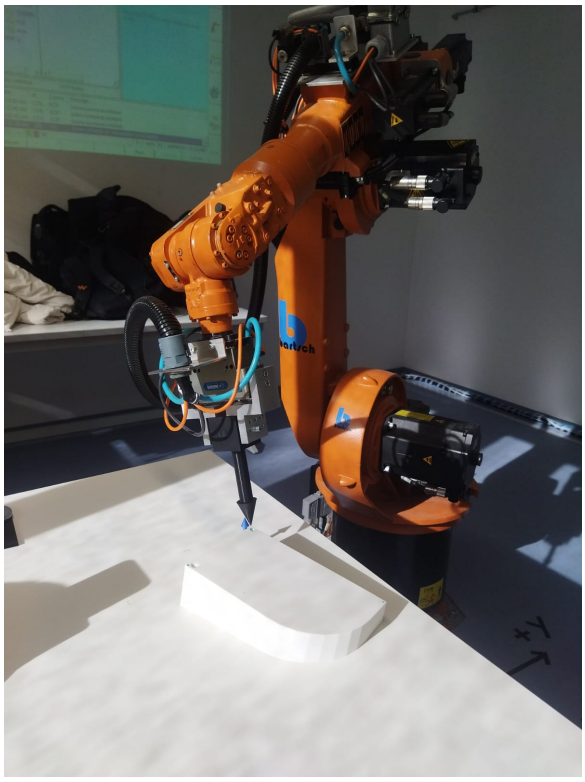


Figure 1.8: First point for base calibration

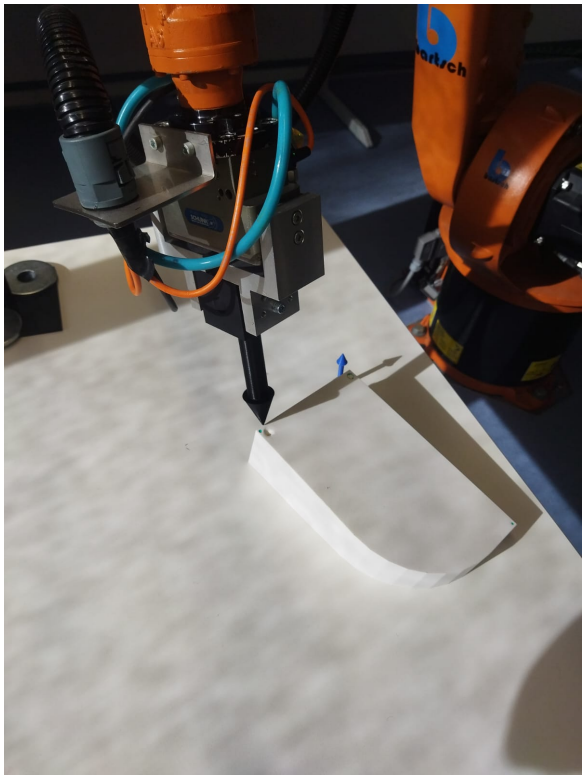


Figure 1.9: Second Point for base calibration

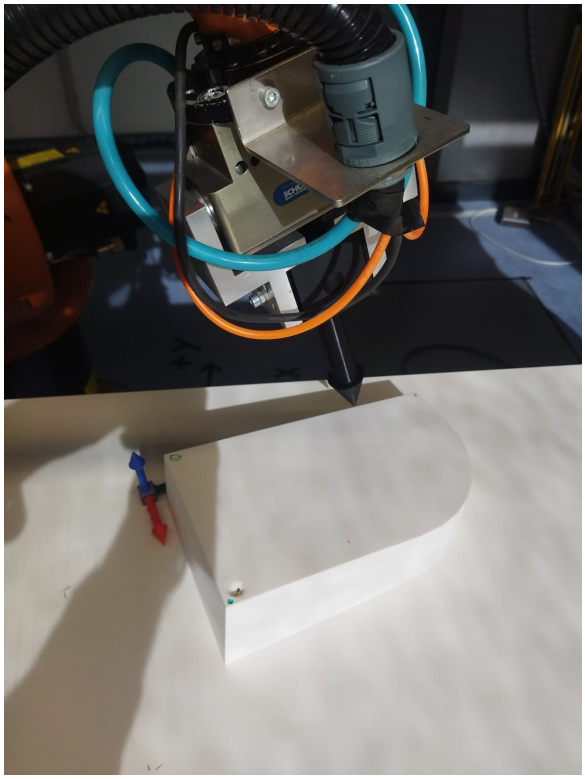


Figure 1.10: Third Point for base calibration

1.5 Create a Box-Following Program

1. First, we defined the tool number and name in the project settings to ensure the correct tool data is used during the program.

2. Then, we taught tool to move the robot along the edges of the box as part of the box-following task.

3. Next,we have moved the tool to the starting position of the box and assign a PTP (Point-to-Point) movement to position the robot quickly and safely at the start.

4. After setting the starting position,we have moved the robot arm to the first corner of the box.

5. At this point,we assigned a LIN (linear) movement type to enable the robot to follow the edges of the box precisely.

6. Then,we move the robot arm to the second corner of the box using a linear movement.

7. Continue by moving the robot to the remaining corners of the box in a similar manner, following the planned path shown below.

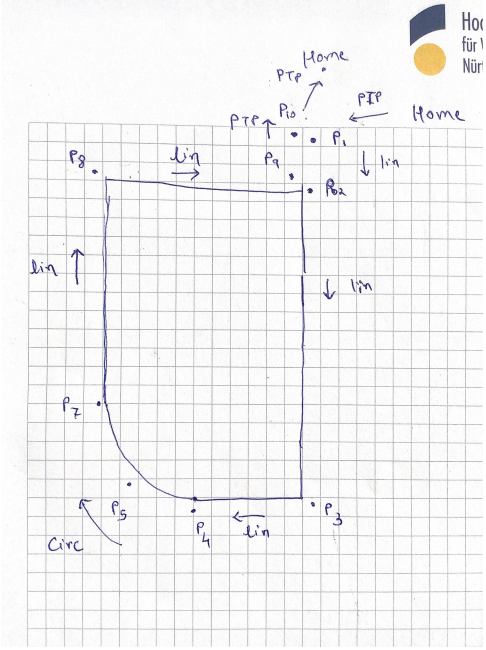


Figure 1.11: Planned path for box-following program

8. Once all corners have been reached,we have moved the robot back to the starting position to complete the path.

9. After completing the path, we have reviewed the entire box-following program structure as shown below.

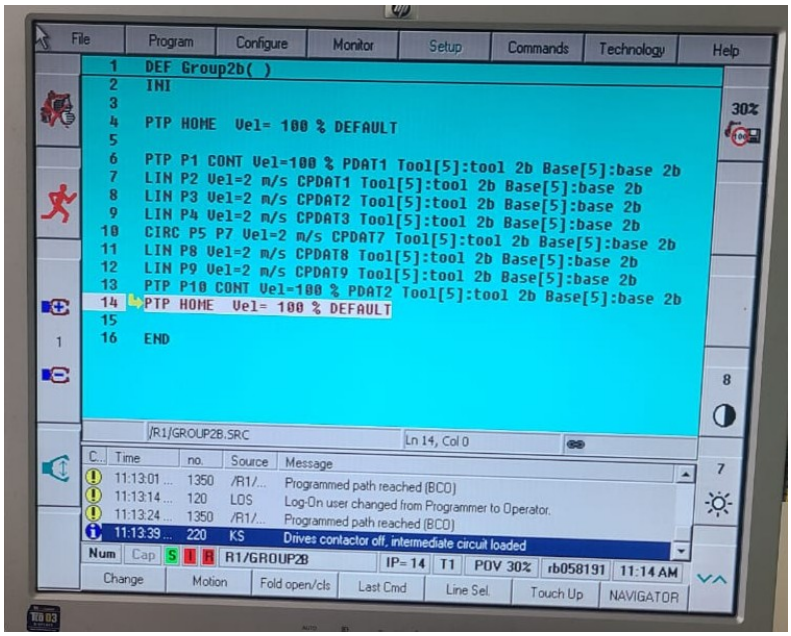


Figure 1.12: Completed box-following program structure

10. Finally, we have saved the program with a meaningful name, such as `grp2b`, for easy identification.

11. Ensure the program logic combines both joint (PTP) and Cartesian (LIN CIRC) movements effectively to achieve accurate and efficient box following.

1.5.1 Re-Teaching the Base

- First, the professor repositioned the box on the workbench.
- Next, we re-taught the base coordinate system by probing the new corners of the box and recording their coordinates.
- Then, we updated the control software by overwriting the old base coordinates with the new ones.
- After that, we ran the existing program to check that the robot correctly followed the edges of the box at its new position.

1.6 Results

1.6.1 Observations

1. The robot moved accurately within all three coordinate systems: **Axes**, **World**, and **Tool**. Each system presented unique advantages and posed specific challenges when controlling the robot's movements.
2. The **tool teaching** process was successful, with the tool tip precisely calibrated to reach all predefined points without error.
3. The **base coordinate system** was accurately defined and calibrated, providing reliable and precise reference points for the robot program.
4. During **program execution**, the robot's tool tip smoothly traced the edges of the box, confirming the program's effectiveness and precision.
5. After **re-teaching the base**, the robot adapted seamlessly to the new box position without requiring any program modifications. This demonstrated the robustness and flexibility of the teaching and calibration processes.

1.7 Conclusion

This laboratory task effectively demonstrated the movement, calibration, and programming of a KUKA robot across different coordinate systems. We successfully performed tool and base teaching, enabling the robot to accurately follow the edges of a box through a custom program. Furthermore, the robot's ability to adapt to changes in the base position without program adjustments highlights the critical importance of precise teaching and calibration in robotic automation. The experiment underlines how careful programming and system configuration can overcome common challenges, ensuring reliable and efficient robotic operation.

Lab Test 2 KUKA.Sim Programming

2.1 Introduction

In this task, we programmed the KUKA robot in KUKA.Sim (A robot simulation tool provieded by KUKA) to draw various shapes, beginning with a basic rectangle and advancing to more intricate figures such as the ”house of Santa Claus.” The goal was to gain hands-on experience with robot motion, tool manipulation, and the application of control structures in programming. We directed the robot’s actions using different coordinate systems and utilized commands like WAIT, WHILE, and IF, with digital inputs controlling the flow of the program.

2.2 Objectives

This task aimed to achieve the following:

1. Understand how to control the robot’s movement by working with multiple coordinate systems.

2. Gain expertise in manipulating and exchanging tools via the robot’s gripper mechanism.

3. Write programs that enable the robot to draw both basic and complex shapes accurately.

4. Employ programming logic commands like `WAIT`, `WHILE`, and `IF` to guide the robot’s movements.

5. Integrate digital inputs to effectively steer the program’s operation and decision-making.

2.3 Methodology

The original simulation file, `_RWU_KukaZelle_9.vcmx`, was duplicated and saved under the new name `_RWU_KUKAZelle_9_Group2a.vcmx`. This copied file was then opened to explore the robot’s movements using different coordinate systems, such as axes, world, and tool coordinates. The robot’s gripper was manually guided along the contour of a white sheet placed in the foreground, with all movement points recorded by the software for further use.

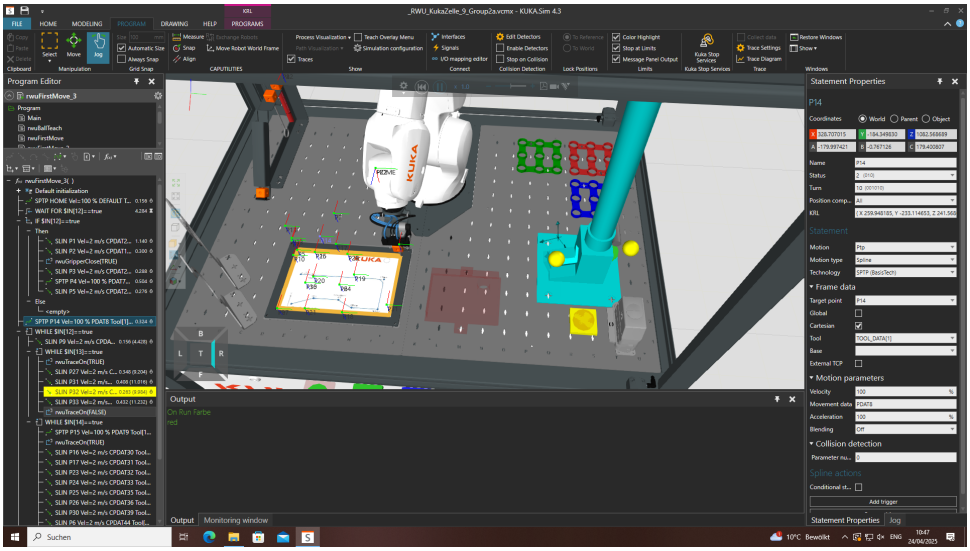


Figure 2.1: Environment

2.3.1 Scribing rectangle using throne

1. A program was created to move the gripper to the thorn located in the tool magazine. The robot approached the thorn using a previously taught point and closed the gripper using the `rwuGripperClose` subroutine.

2. The robot then traveled to the white sheet through intermediate points. The drawing process began with the `rwuTraceOn` command, tracing a simple rectangle based on predefined coordinates. The tracing was stopped by invoking `rwuTraceOff`.

3. After completing the drawing, the robot returned to the thorn holder, released the thorn by setting `rwuGripperClose` to false, and finally moved back to the home position.

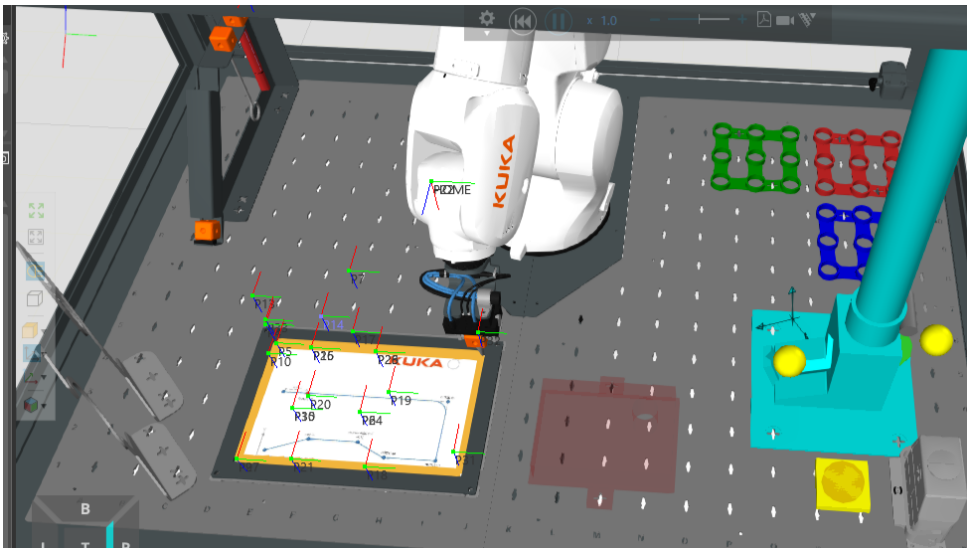


Figure 2.2: Drawing rectangle

2.3.2 Drawing the "House of Santa Claus"

1. The rectangle points were reused to sketch the "house of Santa Claus." The drawing path was planned meticulously to avoid lifting the tool or retracing lines.

2. Programming logic structures such as `WAIT`, `WHILE`, and `IF` were incorporated to control the drawing sequence. Digital inputs `IN[12]`, `IN[13]`, and `IN[14]` were utilized to regulate the flow and execution of different segments of the program.

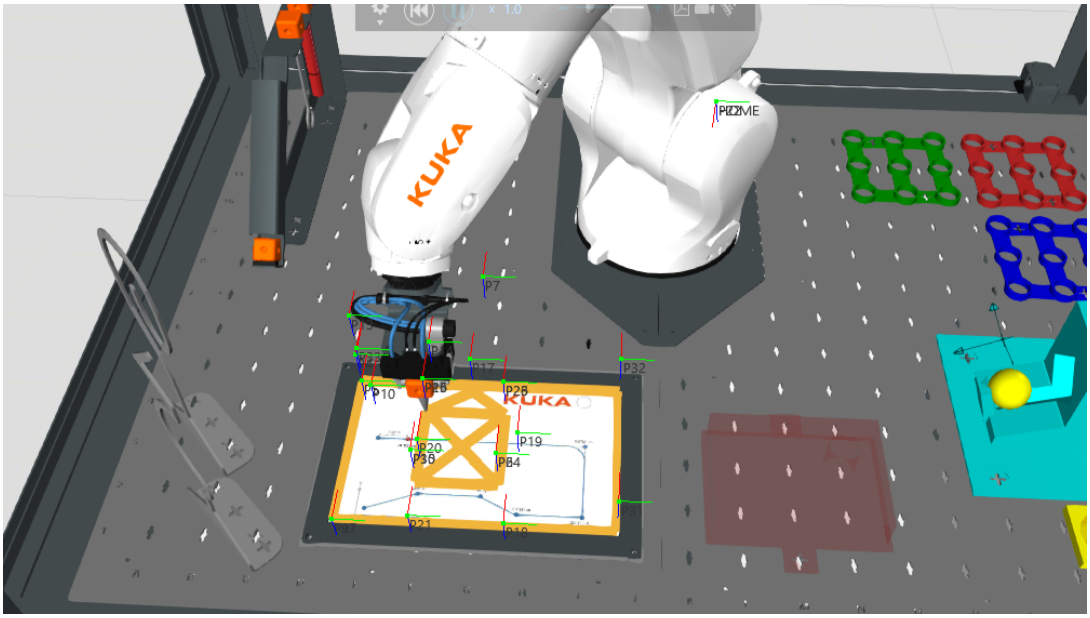


Figure 2.3: Drawing Santa Claus's house

2.4 operating the simulation

- Press Red button to attach the throne
- Press Green button to draw the Rectangular shape
- Press Blue button to draw the House of Santa Claus

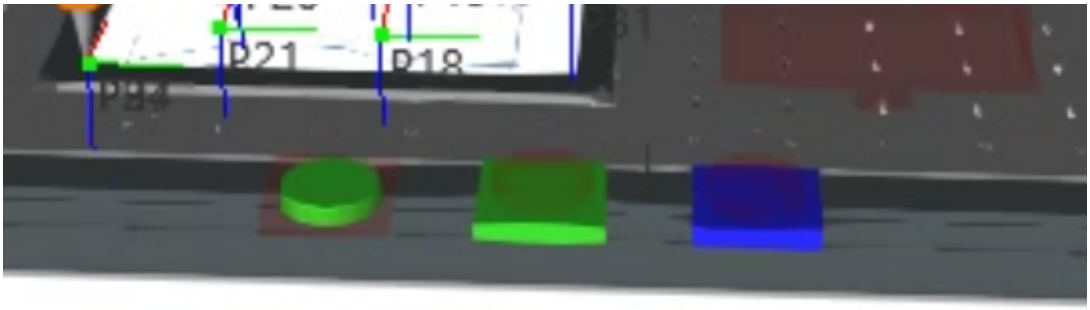


Figure 2.4: Red-Green-Blue Buttons

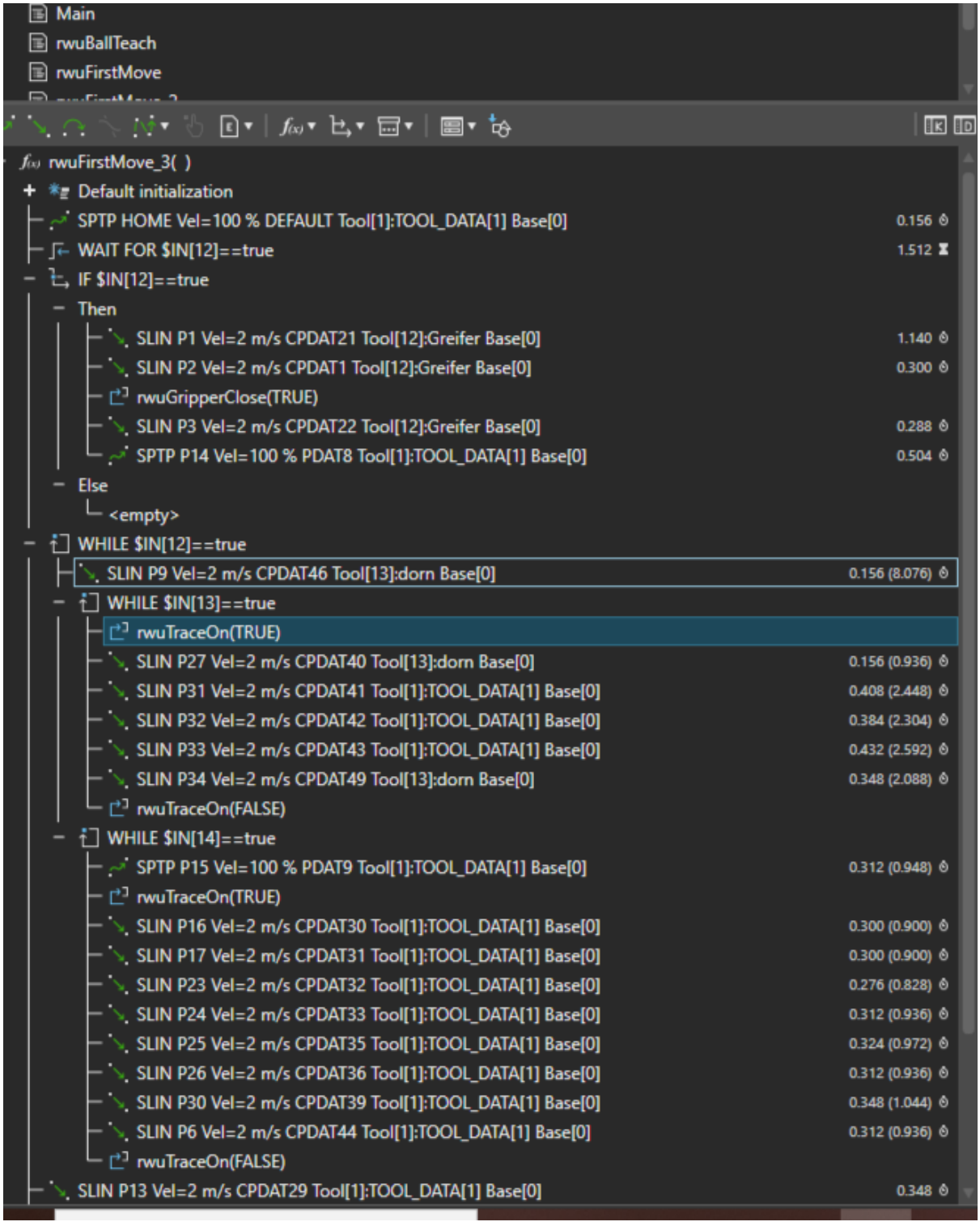


Figure 2.5: Program

2.5 Results

2.5.1 Observations

- The robot precisely traced the perimeter of the white sheet using its open gripper, demonstrating accurate basic movement.
- The thorn was successfully grasped by the robot, enabling it to draw a clean rectangle on the paper as intended.
- The complex figure, known as the "house of Santa Claus," was drawn in a single, continuous motion of eight lines, without lifting the tool or intersecting lines.

2.5.2 Challenges and Solutions

- Achieving precise movement across different coordinate systems initially required trial and error as we familiarized ourselves with how each system influenced the robot's behavior. Regular hands-on adjustments and testing helped us master this.

2.6 Conclusion

This exercise demonstrated the ability to program and simulate the KUKA robot in KUKA.Sim to perform both basic and advanced drawing tasks. We successfully simulated the Robot in software and within KUKA.sim handled tools with precision, and carried out complex drawing operations. The integration of control structures and digital inputs enhanced the flexibility and accuracy of the program. The task highlighted the critical role of careful calibration, thoughtful planning, and structured programming in achieving reliable robotic performance.

Lab Test 3 Sorting Balls by Color using KUKA Robot

3.1 Introduction

In this set of tasks, we focused on simulating a common industrial application using a KUKA robot: sorting balls based on their color. The process consists of picking up balls from a feeder, detecting their color with a sensor, and placing them onto specific pallets according to their color. This experiment covers the simulation phase of the project, while the final implementation and code optimization will be addressed in next experiment.

3.2 Objectives

The main goals of this experiment are as follows:

- **Flow Chart Creation:** Design a comprehensive flow chart to outline the steps of the sorting process.
- **Programming:** Build a robot control program based on the provided framework, incorporating the sorting logic and movement commands.
- **Simulation and Testing:** Run simulations to test and fine-tune the program, ensuring both precision and efficiency.
- **Implementation Readiness:** Prepare the finalized program for deployment on the physical robot for future testing.

3.3 Methodology

3.3.1 Flow Chart Preparation

A detailed flow chart was created to represent each stage of the ball sorting process. The diagram included key steps such as initializing the robot, verifying ball availability, picking up the balls, identifying their color, and placing them onto the correct pallets.

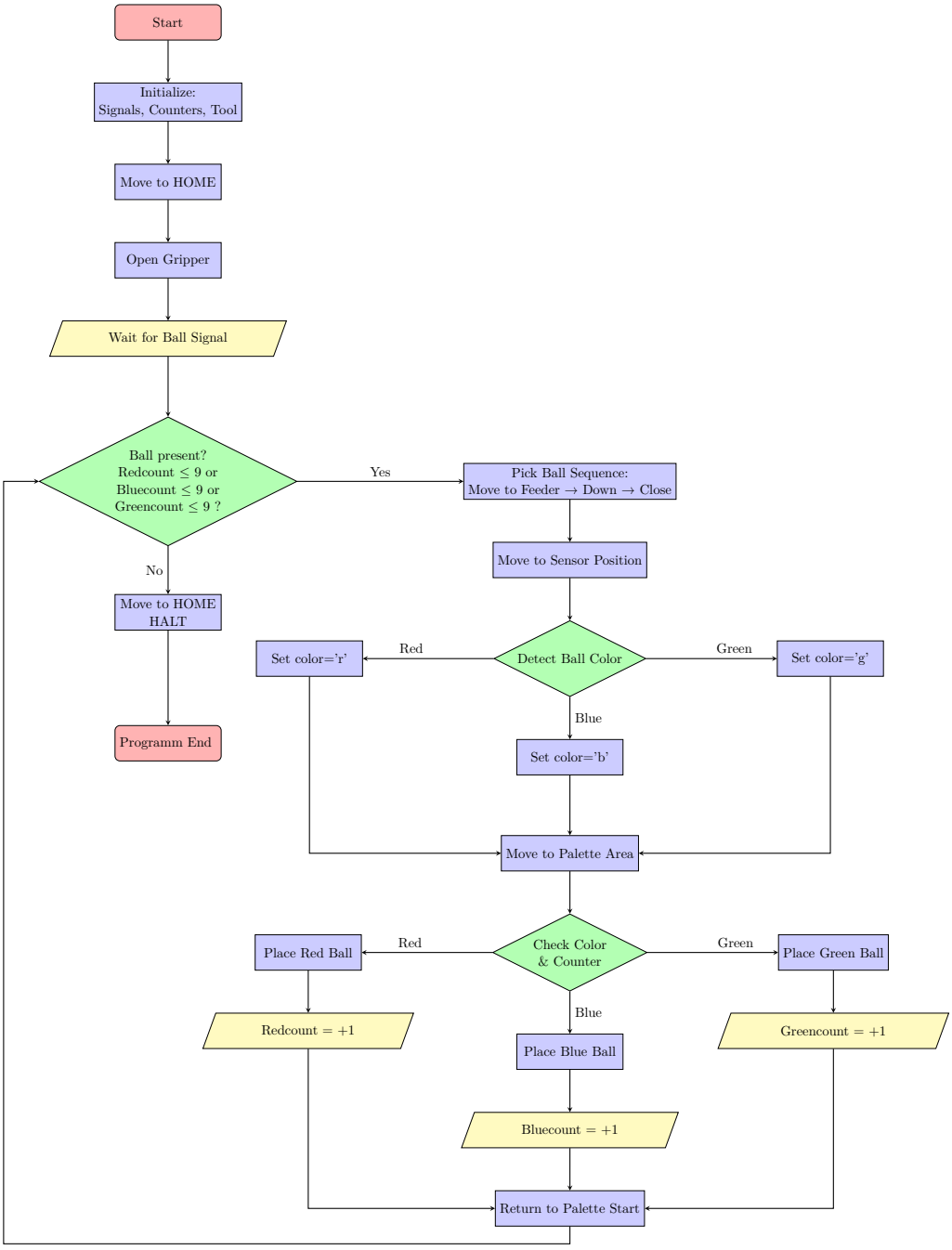
3.3.2 Programming

1. The programming began by duplicating the provided template `rwuBallTeach`. The duplicated program was renamed (for example, `Gr6MoEx3 V1`), with comments added to include the date, group number, and team members' names.

2. Initialization commands were added after the `rwuInitToolBase` subprogram, configuring `Tool[1]` with the specified parameters. The robot's movements and actions were programmed according to the flow chart, using the provided position points.

3. Both PTP and LIN movements were coded with care, ensuring proper gripper operation and integration of sensor readings. Conditional statements (`IF`) were used to handle ball presence and color detection, while `WHILE` loops enabled continuous sorting until set conditions were satisfied. The program ensured that the robot always started and returned to the HOME position at the beginning and end of the sequence.

3.4 Flowchart



Flowchart 3.1: Ball Sorting Program

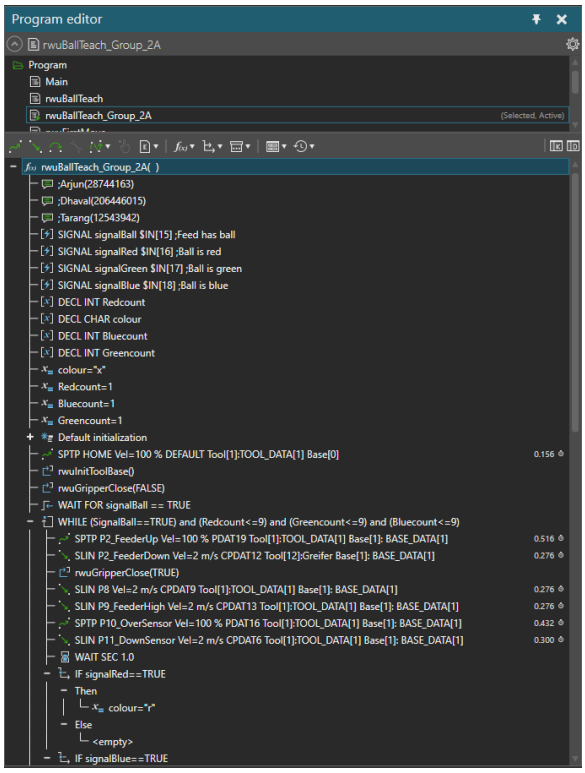


Figure 3.1: Code in KUKA.Sim

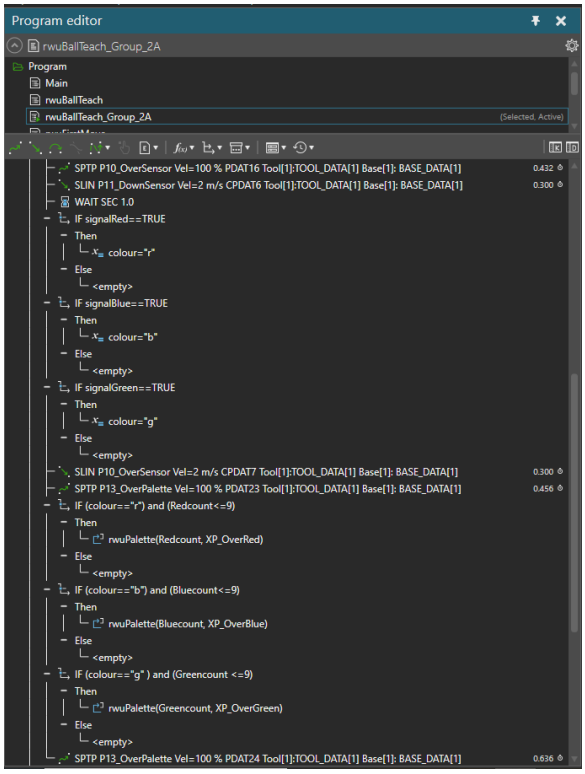


Figure 3.2: Code in KUKA.Sim (cont.)

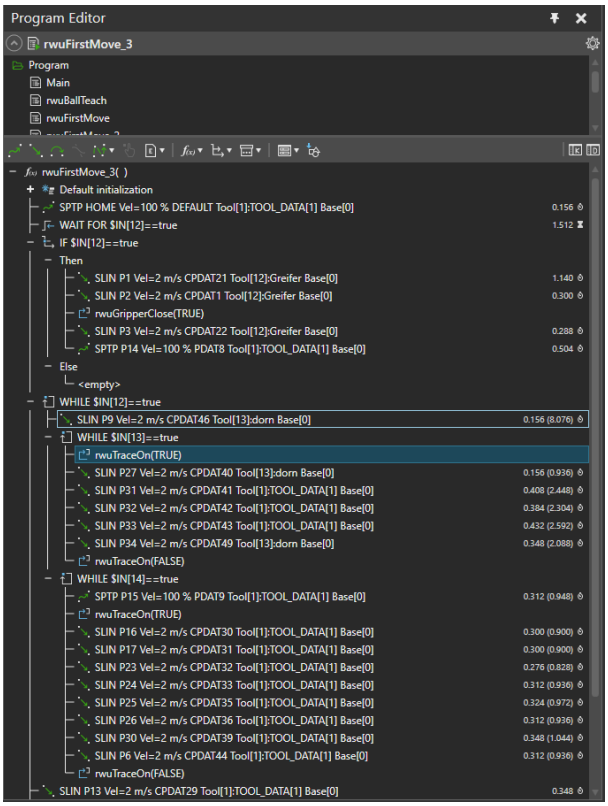


Figure 3.3: Code in KUKA.Sim (cont.)

3.5 code

```
DEF rwuBallTeach_Group_2A()
   ;Arjun(28744163)
   ;Dhaval(206446015)
   ;Tarang(12543942)
   SIGNAL signalBall $IN[15] ;Feed has ball
   SIGNAL signalRed $IN[16] ;Ball is red
   SIGNAL signalGreen $IN[17] ;Ball is green
   SIGNAL signalBlue $IN[18] ;Ball is blue
   DECL INT Redcount
   DECL CHAR colour
   DECL INT Bluecount
   DECL INT Greencount
   colour="x"
   Redcount=1
   Bluecount=1
   Greencount=1
   ;FOLD INI;%{PE}
      ;FOLD BASISTECH INI
         GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS == TRUE DO IR_STOPM()
         INTERRUPT ON 3
         BAS(#INITMOV, 0)
      ;ENDFOLD (BASISTECH INI)
      ;FOLD USER INI
         ;Make your modifications here
      ;ENDFOLD (USER INI)
   ;ENDFOLD (INI)
   SPTP HOME
   rwuInitToolBase()
   rwuGripperClose(FALSE)
   WAIT FOR signalBall == TRUE
   WHILE (SignalBall==TRUE) and (Redcount<=9) and (Greencount<=9) and (Bluecount<=9)
        SPTP XP2_FeederUp
        SLIN XP2_FeederDown
        rwuGripperClose(TRUE)
        SLIN XP8
        SLIN XP9_FeederHigh
        SPTP XP10_OverSensor
        SLIN XP11_DownSensor
        WAIT SEC 1.0
        IF signalRed==TRUE THEN
            colour="r"
        ELSE
        ENDIF
        IF signalBlue==TRUE THEN
             colour="b"
        ELSE
        ENDIF
```

```
      IF signalGreen==TRUE THEN
          colour="g"
      ELSE
      ENDIF

      SLIN XP10_OverSensor
      SPTP XP13_OverPalette

    IF (colour=="r") and (Redcount<=9) THEN
       rwuPalette(Redcount, XP_OverRed)
    ELSE
    ENDIF
    IF (colour=="b") and (Bluecount<=9) THEN
       rwuPalette(Bluecount, XP_OverBlue)
    ELSE
    ENDIF
    IF (colour=="g" ) and (Greencount <=9) THEN
       rwuPalette(Greencount, XP_OverGreen)
    ELSE
    ENDIF
      SPTP XP13_OverPalette
ENDWHILE
;FOLD SPTP HOME
SPTP XHOME
HALT
HALT
HALT
```

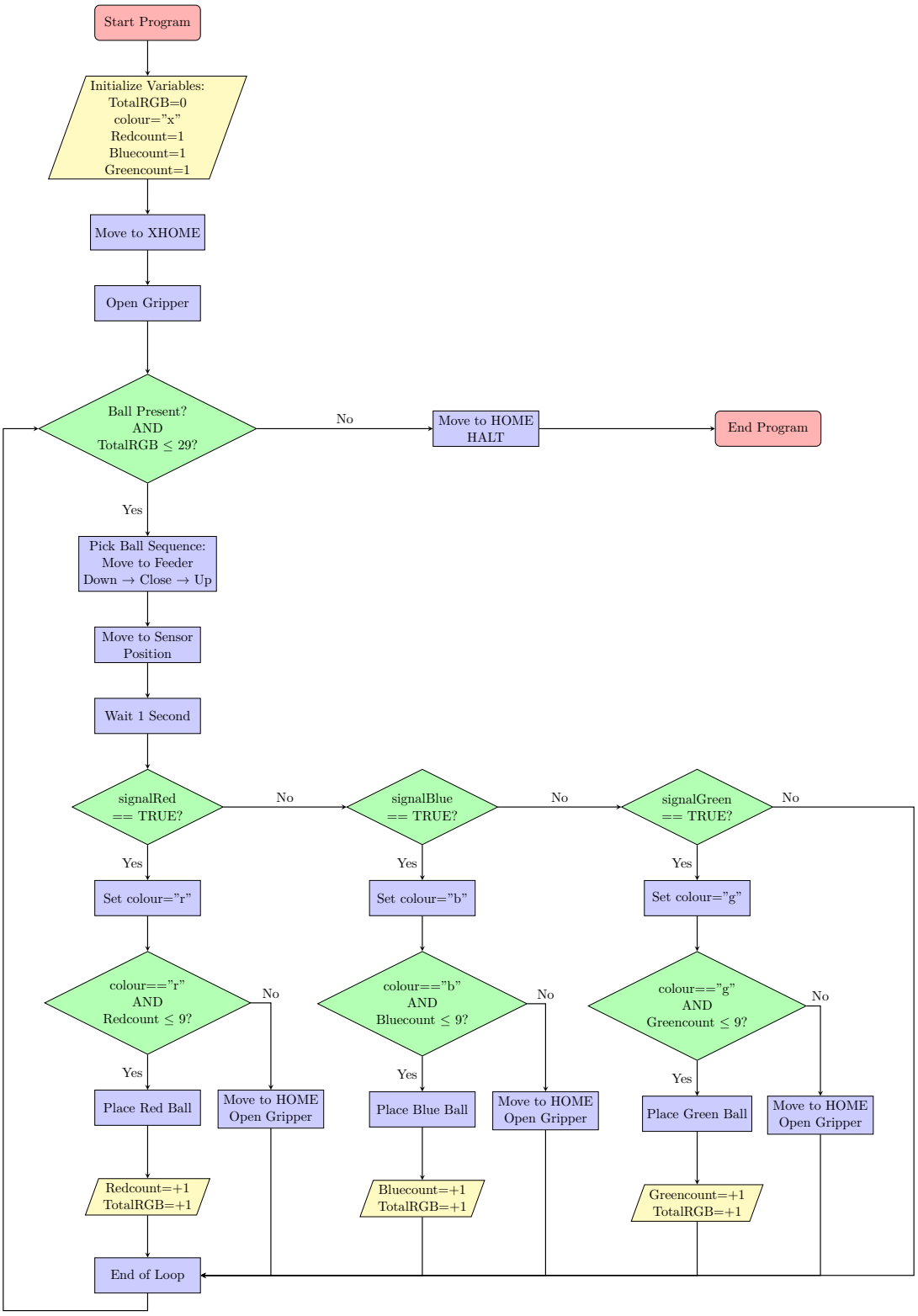
3.6 Testing

For testing, We started by running simulation to check how well the program worked and to observe how the robot moved through the sorting process. As we tested, we made small adjustments to the code wherever improvements were needed to ensure smoother and more efficient operation. Once we were satisfied with the performance, we exported the `.src` and `.dat` files so they would be ready for use in the next task, where the program will be implemented on the actual robot.

3.7 Extended version

- In the previous version, the robot returned to the home position as soon as **any one pallet became full**, interrupting the sorting process.
- In the updated version, we allow the robot to continue operating even if one pallet is full.
- When a pallet became full, the robot:
  - **Does not stop**, but instead checks the color of the next incoming ball.
  - If the detected ball's color corresponds to a **non-full pallet**, the robot proceeds with **normal placement**.
  - If the ball's color matches the **full pallet**, the robot **discards the ball** by moving to the home position and opening the gripper.
- This updated strategy ensures that the robot continues sorting until **all three pallets are full**, reducing unnecessary halts and improving overall **system efficiency**.

3.8 Flowchart for Ball Sorting Program



Flowchart 3.2: Extended version of Ball Sorting Program

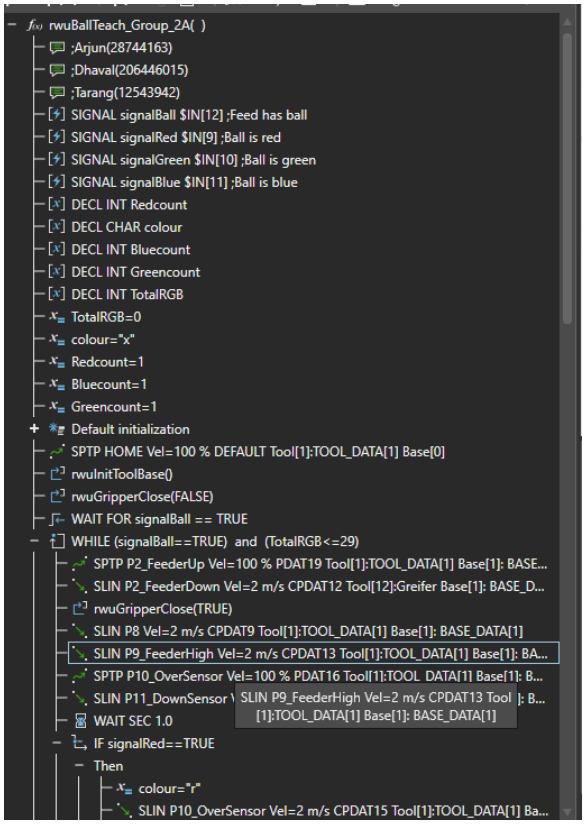


Figure 3.4: Ext. Ver. Code in KUKA.Sim

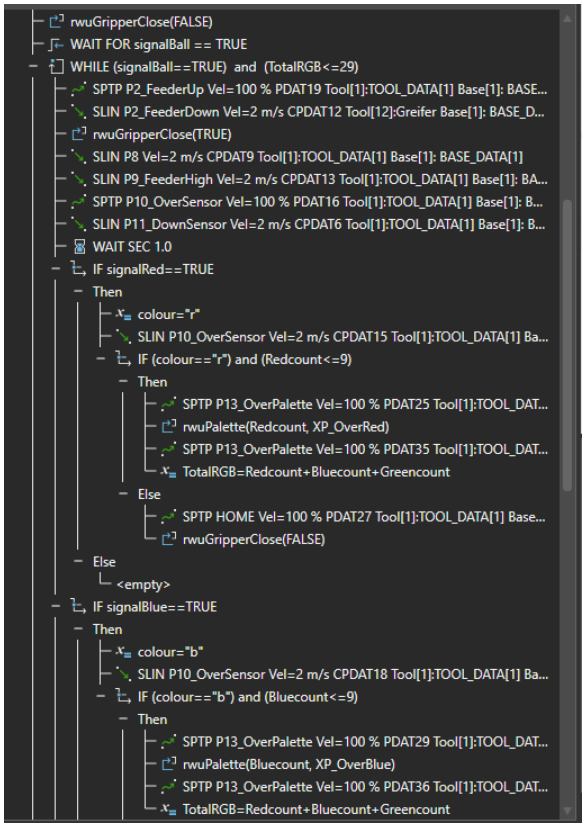


Figure 3.5: Ext. Ver. Code in KUKA.Sim (cont.)

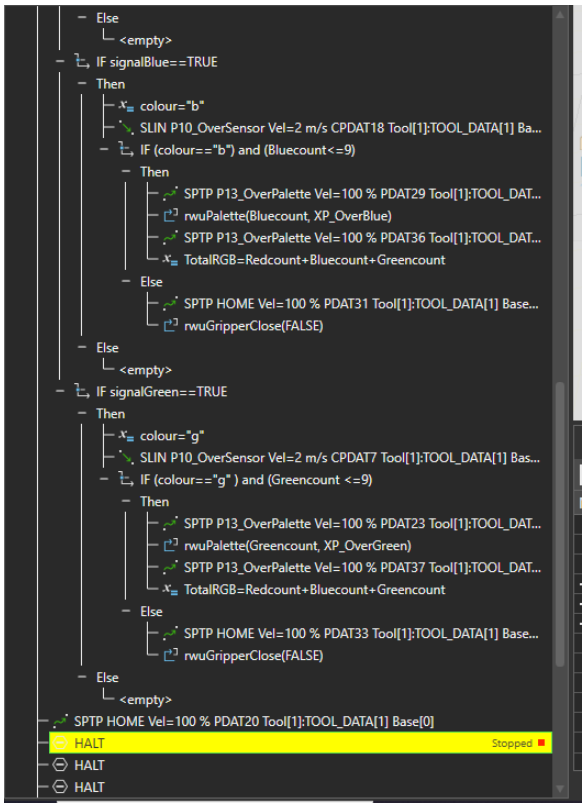


Figure 3.6: Ext. Ver. Code in KUKA.Sim (cont.)

3.9 Code

```
 DEF rwuBallTeach_Group_2A()
;Arjun(28744163)
;Dhaval(206446015)
;Tarang(12543942)
SIGNAL signalBall $IN[15] ;Feed has ball
SIGNAL signalRed $IN[16] ;Ball is red
SIGNAL signalGreen $IN[17] ;Ball is green
SIGNAL signalBlue $IN[18] ;Ball is blue
DECL INT Redcount
DECL CHAR colour
DECL INT Bluecount
DECL INT Greencount
DECL INT TotalRGB
TotalRGB=0
colour="x"
Redcount=1
Bluecount=1
Greencount=1
;FOLD INI;%{PE}
   ;FOLD BASISTECH INI
      GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS == TRUE DO IR_STOPM()
      INTERRUPT ON 3
      BAS(#INITMOV, 0)
   ;ENDFOLD (BASISTECH INI)
   ;FOLD USER INI
      ;Make your modifications here
   ;ENDFOLD (USER INI)
;ENDFOLD (INI)
SPTP XHOME
rwuInitToolBase()
rwuGripperClose(FALSE)
WAIT FOR signalBall == TRUE
WHILE (signalBall == TRUE) AND (TotalRGB <= 29)
   SPTP XP2_FeederUp
   SLIN XP2_FeederDown
   rwuGripperClose(TRUE)
   SLIN XP2_FeederUp
   SLIN XP9_FeederHigh
   SPTP XP10_OverSensor
   SLIN XP11_DownSensor
   WAIT SEC 1.0
   IF (signalRed == TRUE)  THEN
      colour="r"
      SLIN XP10_OverSensor
      IF (colour == "r") AND (Redcount <= 9) THEN
         SPTP XP13_OverPalette
         rwuPalette(Redcount, XP_OverRed)
```

```
            SPTP XP13_OverPalette
            TotalRGB=Redcount + Bluecount + Greencount
         ELSE
            SPTP XHOME
            rwuGripperClose(FALSE)
         ENDIF
      ELSE
      ENDIF
      IF signalBlue == TRUE THEN
         colour="b"
         SLIN XP10_OverSensor
         IF (colour == "b") AND (Bluecount <= 9) THEN
            SPTP XP13_OverPalette
            rwuPalette(Bluecount, XP_OverBlue)
            SPTP XP13_OverPalette
            TotalRGB=Redcount + Bluecount + Greencount
         ELSE
            SPTP XHOME
            rwuGripperClose(FALSE)
         ENDIF
      ELSE
      ENDIF
      IF signalGreen == TRUE THEN
         colour="g"
         SLIN XP10_OverSensor
         IF (colour == "g") AND (Greencount <= 9) THEN
            SPTP XP13_OverPalette
            rwuPalette(Greencount, XP_OverGreen)
            SPTP XP13_OverPalette
            TotalRGB=Redcount + Bluecount + Greencount
         ELSE
            SPTP XHOME
            rwuGripperClose(FALSE)
         ENDIF
      ELSE
      ENDIF
   ENDWHILE
   SPTP XHOME
   HALT
   HALT
   HALT
```

3.10 Testing of extended version

For testing, we initially ran simulations to evaluate the correctness and performance of the updated sorting logic. The goal was to verify whether the robot could correctly detect ball colors, track individual pallet counts, and handle full pallet conditions as intended. During simulation, we carefully observed the robot's movements and decision-making at each stage of the process.

Throughout testing, we identified and refined specific parts of the code to enhance the robot's behavior—particularly in handling cases where one or more pallets were full. These iterative improvements helped ensure the robot continued sorting efficiently without unnecessary interruptions. After confirming the program's stability and accuracy in the simulated environment, we exported the final version of the `.src` and `.dat` files. These files are now ready for deployment in the next phase, where the program will be transferred to the physical robot for real-world implementation.

3.11 Results

3.11.1 Observations

- A detailed flowchart outlining the complete ball sorting logic was successfully designed and approved.
- The program was developed and implemented according to the provided framework and predefined coordinates.
- Simulations confirmed that the robot accurately sorted balls by color with smooth and coordinated movements.
- Logical improvements, such as handling full pallets more efficiently, were integrated and tested successfully.

3.12 Conclusion

This task demonstrated the successful development and simulation of a ball-sorting program for a KUKA robot. The process began with designing a flowchart to visualize the control logic, followed by structured implementation and iterative improvements during simulation testing. Key functionalities, including accurate color detection, pallet tracking, and handling of full pallet conditions, were validated in a controlled virtual environment. The refined logic not only improved operational efficiency but also ensured continuous processing until all pallets reached their limits. Overall, the results of this simulation provide a strong foundation for deploying the program on the physical robot in the next phase of the project.

Lab Test 4 Implementation of Ball Sorting Program on Robot

4.1 Introduction

In this phase, we moved from simulation to practical implementation by deploying the ball-sorting program on an actual KUKA robot. This section covers the final testing stages, including position verification and validation of the robot's performance to ensure precise and smooth sorting operations at progressively higher speeds.

4.2 Objectives

- Prepare and adapt the program which was developed during the simulation phase for real-robot use.
- Verifying robot points to ensure precise and accurate movements.
- Conducting incremental testing starting at low speed and gradually increasing to normal operational speed.
- Validate all program functions and obtain final approval from the lab supervisor.

4.3 Methodology

4.3.1 Programming

- First, we copied the provided program template, `rwuBallTeach`, and renamed it, adding the current date, group number, and member names as comments. We then inserted the program after the `rwuInitToolBase` subroutine.
- Next, we implemented the movements and actions according to the flowchart, ensuring that both PTP and LIN motions were correctly programmed using the provided points.
- Additionally, we incorporated key functions such as `rwuGripperClose` for gripping and `rwuPalette` for palletizing. We also used IF statements to verify ball availability and detect colors accurately.

- Finally, we employed WHILE loops to continuously sort balls until the feeder was empty or all pallets were full. Throughout the process, we ensured the program started and ended with the robot positioned at HOME for safety and consistency.

4.3.2 Validation

- We presented the finalized program to the lab supervisor for evaluation and approval.
- After approval, we exported the `.src` and `.dat` files to prepare for real-robot implementation.
- We saved the source files for documentation purposes and future reference.

4.4 Real-Robot Implementation

4.4.1 Preparing the Code

1. **Loading the Program:**
   - We have successfully transferred the program files (`*.src` and `*.dat`) from the simulation environment to the PC.
   - Afterwards, We have configured the digital inputs as follows:

Signal	Digital Input (DI)
Red ball	9
Green ball	10
Blue ball	11
Ball availability signal	12

Table 4.1: Digital Input Configuration for Ball Detection

   - Lastly, We provided the program to the lab supervisor to load onto the robot.

4.4.2 Testing and Adapting the Code

1. **Initial Setup:**
   - First, We confirmed the robot was positioned at HOME before starting.
   - then, We thoroughly checked and verified all connections and sensor configurations.
2. **Points Verification:**
   - In this, We moved the robot sequentially through all taught points using specific commands such as `PTP HOME`.

- The robot began testing from the HOME position.

3. **Incremental Speed Testing:**

   - We started the program testing at low speed and gradually increased it.
   - Then, We closely monitored the robot's movements to maintain safe operation.
   - afterwards, We verified the proper functioning of the gripper, sensors, and program termination conditions.

4. **Final Validation:**

   - We presented the fully tested and validated program to the lab supervisor for approval.

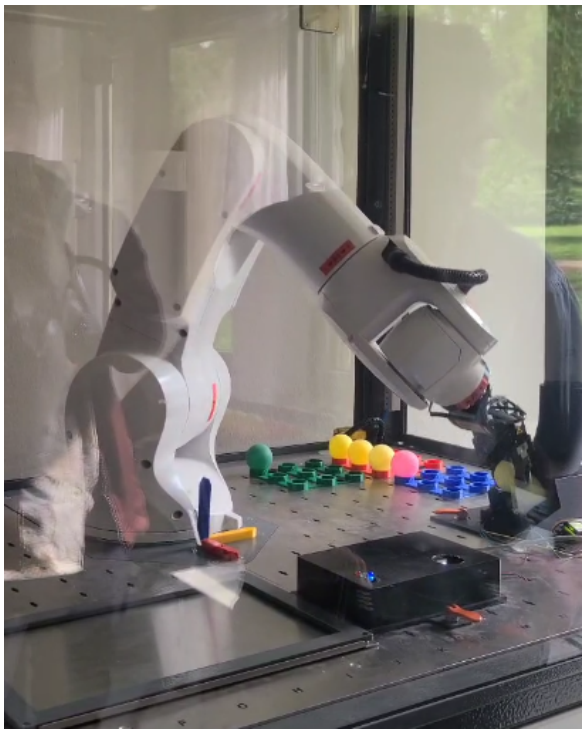


Figure 4.1: Peaking up the ball

Figure 4.2: Color detection on the sensor

Figure 4.3: Move on tho the pallet

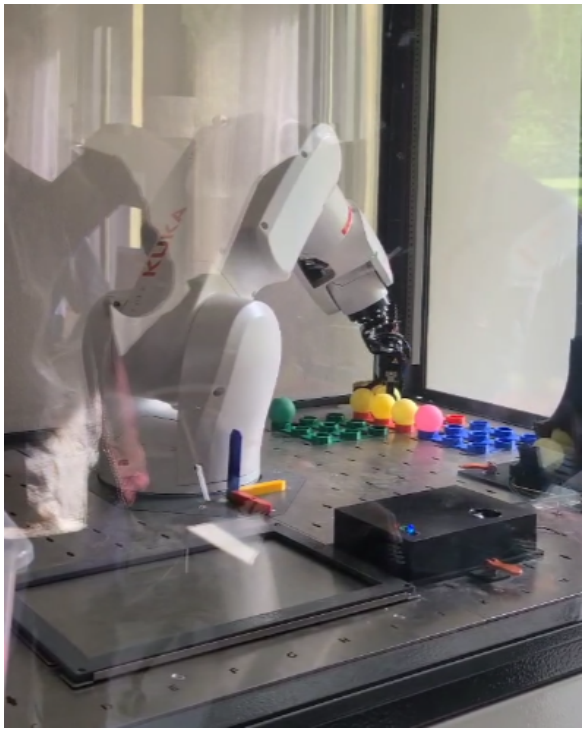


Figure 4.4: Placing ball on the right pallet

Figure 4.5: Home position after one pallet full

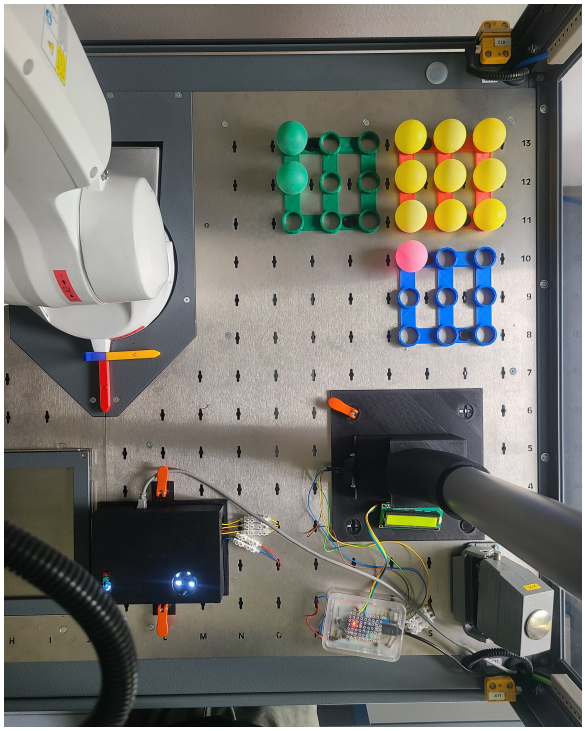


Figure 4.6: Pallets

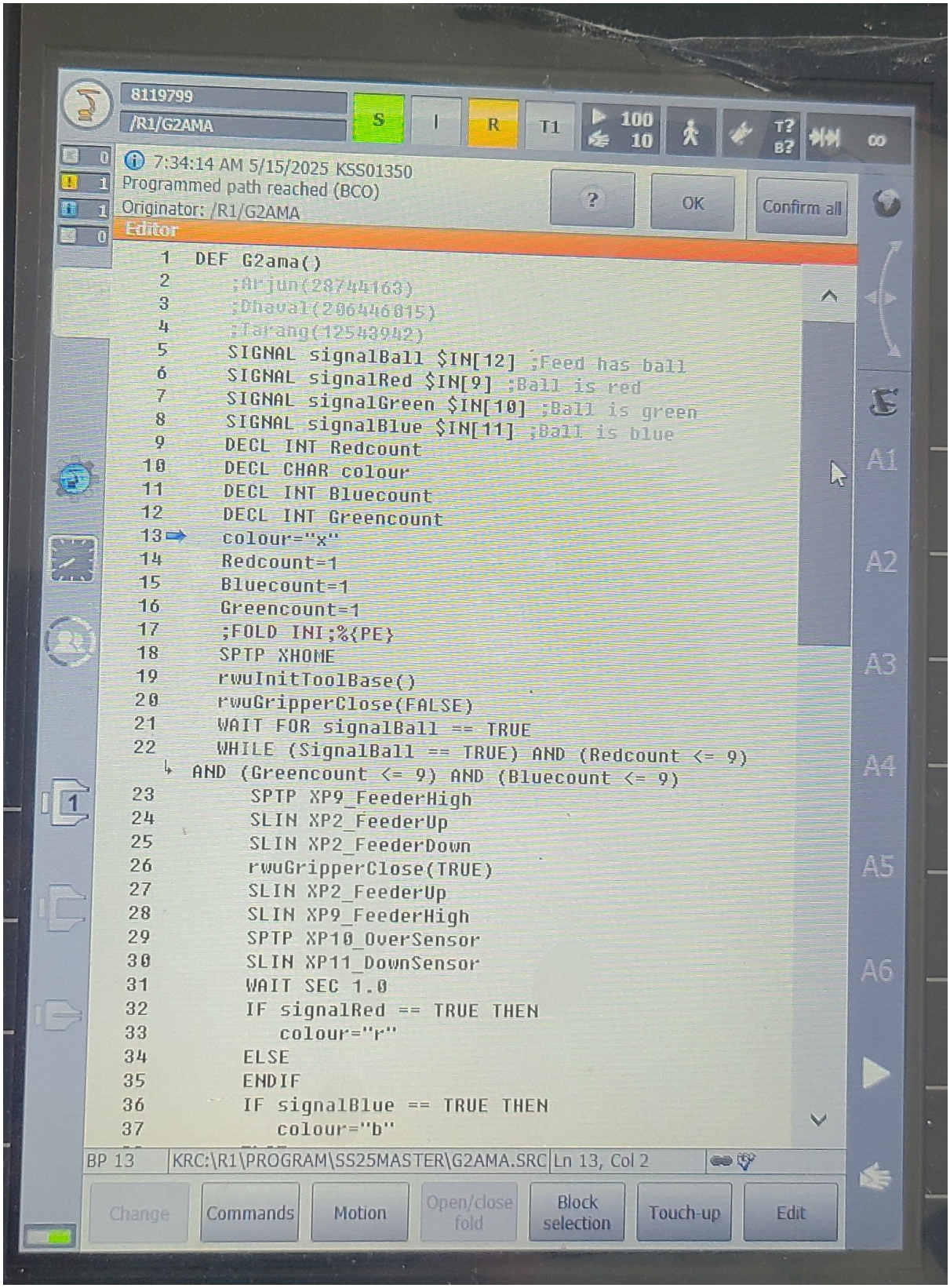


Figure 4.7: Program in Controller

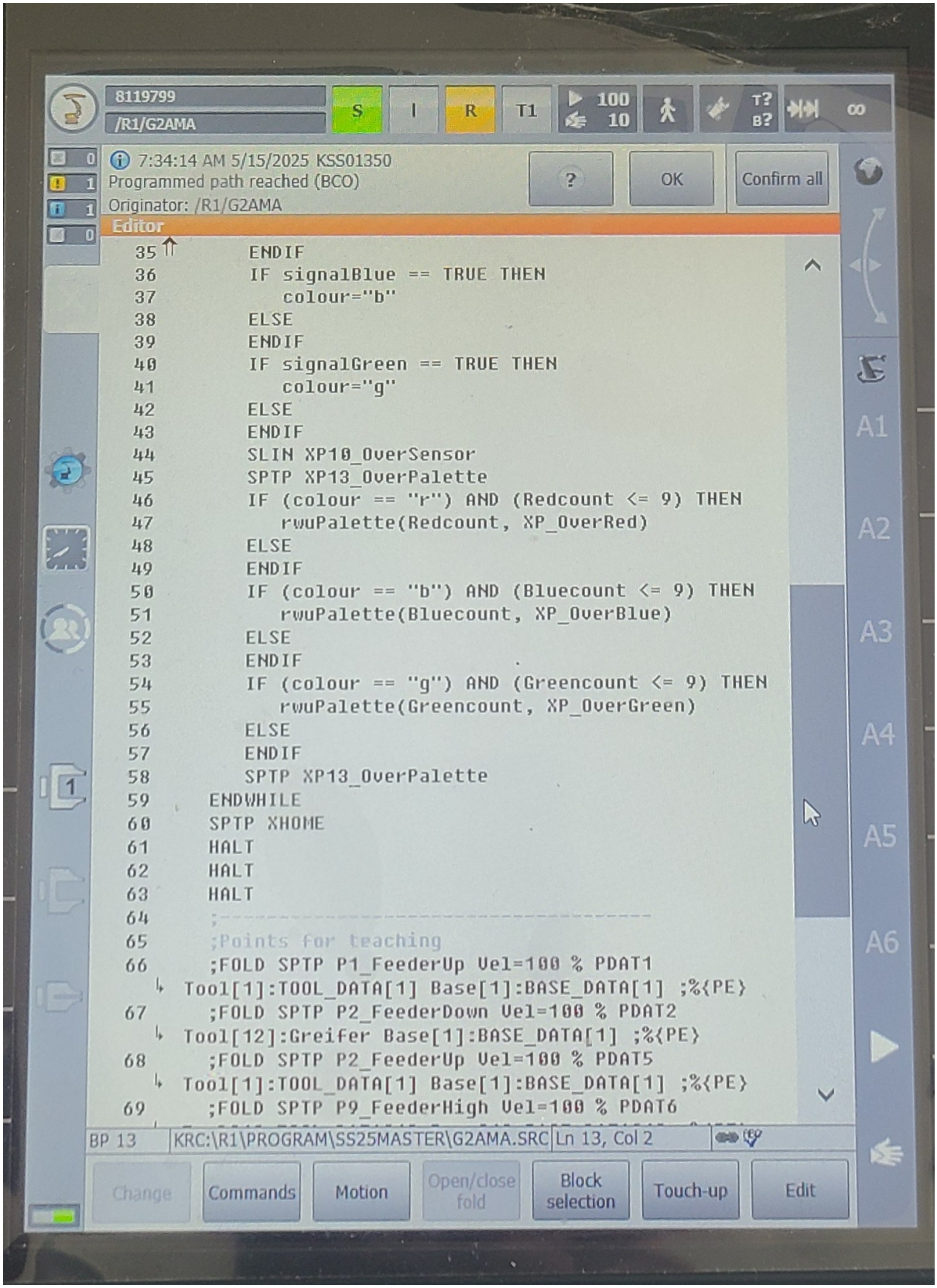


Figure 4.8: Program in Controller (cont.)

4.5 Results

4.5.1 Observations

- We successfully verified and fine-tuned all the taught points to ensure accuracy.
- The program ran smoothly at low speeds and maintained reliable performance as the speed increased.
- All components, including the gripper and sensors, functioned properly throughout the tests.
- The robot was successfully initialized and positioned at HOME.
- It accurately detected and picked up balls from the feeder.
- The ball colors were correctly identified and recorded.
- Balls were placed onto their respective pallets according to color.
- Upon task completion or when pallets reached full capacity, the robot returned to the HOME position.

4.6 Conclusion

This task effectively showcased the transition from simulation to actual robot deployment for the ball sorting operation. Careful validation of taught positions combined with gradual speed testing guaranteed precise and safe robot movements. The successful execution of this phase lays a strong foundation for further program enhancements and optimization geared toward industrial use.