EXPERIMENT KW15: Robotic Arm Control and Target Positioning using Color Recognition

Related course: KIE3010 (Instrumentation)

OBJECTIVES:

1. To understand the working principles of an intelligent transport system using a robotic arm and line-following car.
2. To explore ROS (Robot Operating System) commands to control the robot and manage the intelligent transport game.

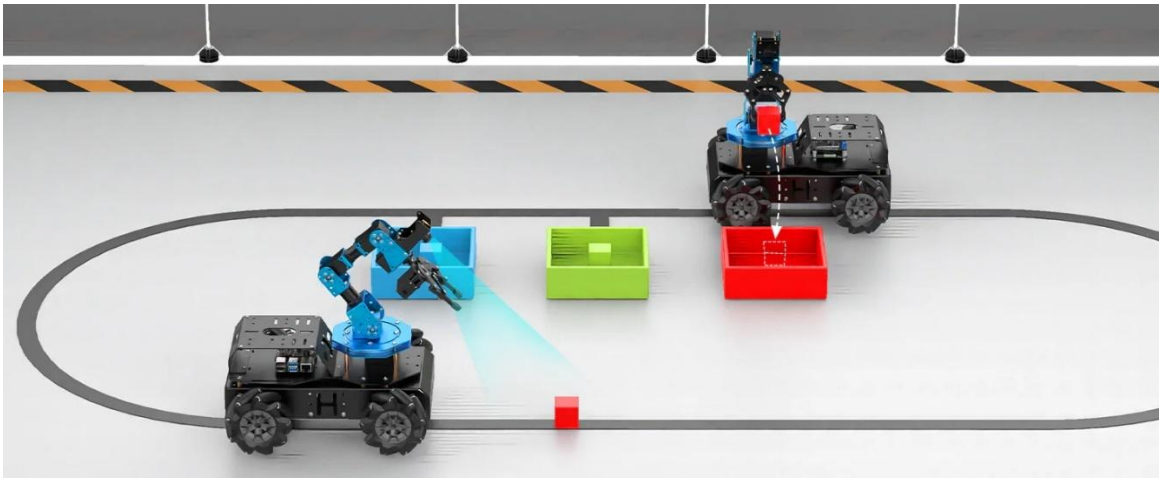
EQUIPMENT:

ArmPi Pro Robotic Kit, PC/Laptop with Wi-Fi connection, Coloured blocks, Tape

REFERENCE(S):

Software and tutorials of ArmPi Pro can be found from
https://drive.google.com/drive/folders/1UNf2xl2Lc8w-Axk9q2BSJ8_6pFhUJMJ0?usp=drive_link

INTRODUCTION:



In this experiment, place a coloured block within the recognizable range of the camera. Once the ArmPi Pro recognizes the block's colour, it will pick the block up, start line-following driving, and place the block in its corresponding position.

PROCEDURES:

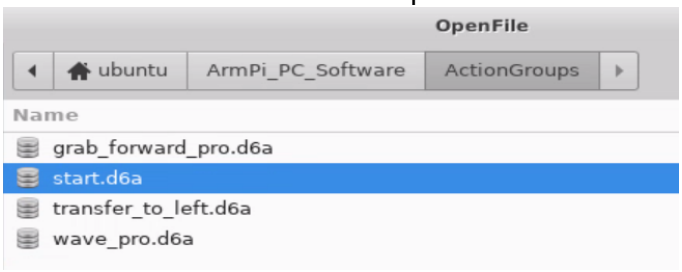
Part 1: Adjusting Servos Based on Target Action

Action editing refers to adjusting the angles of corresponding servos based on the target action to achieve the action, and multiple actions are combined into the action group. In this experiment, we are going to create an action group to allow robotic arm to transport the block to the left side.

1. Switch on Raspberry Pi expansion board. On your computer, connect to Raspberry Pi through LAN connection (refer to steps in KW14).
2. Double-click  to open PC software.



3. Click “Open action file” and select “start.d6a” action group file in `/home/ubuntu/ArmPi_PC_Software/ActionGroups`. Then click “Open” to import this action group into action date list to set an initial posture for ArmPi Pro.



4. Click ▶ in front of No.1 action to update the servo angle information in servo control area.
5. Drag the bar slider of ID:5 servo to “500” to allow robotic arm to move down to the block.
6. Set the running time as 800ms and click “Add action” to get the second action.



7. To make action group smoother, add a transition action after setting an action and set the running time as 100ms. Then click “Add action” to get No.3 action.
8. Repeat the steps as shown below to add all actions.

Index	Time	ID:1	ID:2	ID:3	ID:4	ID:5	ID:6
1	800	500	500	300	900	700	500
2	800	500	500	300	900	500	500
3	100	500	500	300	900	500	500
4	500	150	500	300	900	450	500
5	200	150	500	300	900	450	500
6	600	150	500	300	900	370	500
7	200	150	500	300	900	370	500
8	600	600	500	300	900	370	500
9	700	600	500	300	700	500	500
10	200	600	500	300	700	500	500
11	800	600	500	300	700	500	750
12	200	600	500	300	700	500	750
13	800	600	500	400	850	350	750
14	100	600	500	400	850	350	750
15	400	100	500	400	850	350	750
16	700	500	500	300	700	500	750
17	100	500	500	300	700	500	750
18	1000	500	500	300	900	700	500

9. Next, let's look at the performance. Select No.1 action and click "Run". You should see how the robotic arm is controlled to transport the block to the left side. If you want to execute this action cyclically, you can check "Loop".

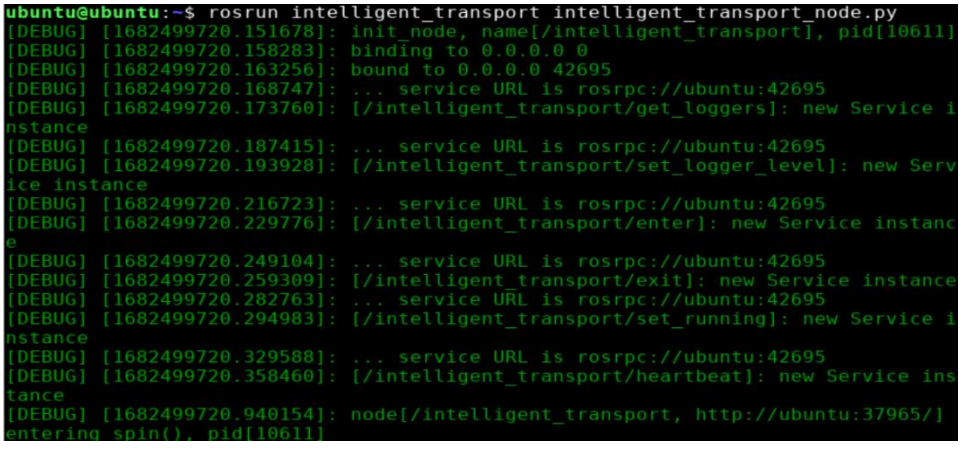
Part 2: Intelligent Transport

The source codes of program are located in the following files:

- Image Processing:
  /home/ubuntu/armpi_pro/src/visual_processing/scripts/visual_processing_node.py
- Function Implementation:
  /home/ubuntu/armpi_pro/src/intelligent_transport/scripts/intelligent_transport_node.py

1. Locate and open the source code of the Function Implementation program. Before starting the experiment, replace the existing code with the contents of the file located at:
   /home/ubuntu/Desktop/Code/ intelligent_transport_node.py
2. Click Applications in the lower left corner and select Terminal Emulator to enter the terminal.
3. Enter the following command to start the game:

```
rosrun intelligent_transport intelligent_transport_node.py
```



4. Do not close the opened terminal and open a new terminal. Enter the following command to enter the game:

```
rosservice call /intelligent_transport/enter "{}"
```

5. Start image transmission using a browser (eg. Google Chrome). Enter the default IP address 192.168.149.1:8080/. Then, click the option shown in the following figure to open the display window of the transmitted image.



6. Analyse the code to understand:
   - **Block Gripping**: How the program detects and grips the block.
   - **Line Following**: How the program follows the designated line.
   - **Placement Line Recognition**: How the program recognizes placement lines.
   - **Block Placement**: How the program places the block in the correct position.

   Refer to the appendices to understand how each function is realized.
7. Now, enter the terminal and run the game by entering the command:
   ```
   rosservice call /intelligent_transport/set_running "data: true"
   ```
8. Observe the robot gripping the block, following the line, and placing the block. Record a video and include it in your lab report.

Stop and Exit the Game

1. Enter the following command to stop the game:
   ```
   rosservice call /intelligent_transport/set_running "data: false"
   ```
2. Enter the following command to exit the game:
   ```
   rosservice call /intelligent_transport/exit "{}"
   ```
3. Shut down the Raspberry Pi properly by using the Shutdown or Logout option in the Applications menu.
4. Switch off the Raspberry Pi expansion board physically to reset all hardware and software states.

OPEN-ENDED TASKS

Research real-world applications of intelligent transport systems in industries (e.g., warehouse automation, manufacturing).

- Analyse the challenges such as Navigation & Path Planning Issues, Sensor Limitations, Efficiency Concerns and Error Handling.
- Propose two realistic improvements for the ArmPi Pro based on real-world applications. Justify how these modifications could enhance performance and applicability.

INSTRUCTIONS FOR LAB REPORT SUBMISSION OF KW14 AND KW15

Screenshot the code snippets/settings you modified during the experiments. Include these in the report with brief captions explaining the changes.

KW14

- Present all data from Experiment KW14 (e.g., time taken, speed for red, green, blue tapes at default, 50, and 150 velocities) in clear, numbered tables with units (m, s, m/s). Include averages for each colour and velocity setting.
- Include at least one graph summarizing KW14 results (Label axes, include units).
- Perform discussions as instructed in the lab sheet.

KW15

- Provide a link to the recorded video of KW15 (e.g., Google Drive, YouTube). Describe key observations: success/failure of block gripping, line-following smoothness, placement accuracy, and any errors (e.g., missed lines).
- Complete the open-ended tasks.

APPENDIX 1: PROGRAM ANALYSIS

Grip the Block

The robot uses inverse kinematics to calculate the position of the robotic arm for picking and placing blocks. The pitch angle is calculated based on the given coordinates, pitch angle and the range of pitch angle, as follows:

```
target = ik.setPitchRange((0, round(y_dis, 4), 0.03), -180, -180, 0)
```

- position of the target on x-axis
- position of the target on y-axis
- position of the target on z-axis
- pitch angle
- range of pitch angle

The robot controls the servos to move the robotic arm and gripper.

`bus_servo_control.set_servos(joints_pub, 20, ( (3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)))`

- The first parameter `joints_pub` is to publish the message of servo control node.
- The second parameter `20` is the runtime.
- The third parameter `((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis))` represents the servo IDs and their corresponding angles.

Line Following

After gripping the block, the car will follow the line. Firstly, judge if there is the line within the detected range. The code is shown in the following figure:

```
elif line_width > 0:
```

Then the current x-coordinate of line subtracts the value of ideal centre point. Get the yaw rate by PID mapping to adjust the speed of motor.

```
if abs(line_center_x - img_w/2) < 30:
    line_center_x = img_w/2
line_x_pid.SetPoint = img_w/2
line_x_pid.update(line_center_x)
dx = round(line_x_pid.output, 2)
dx = 0.8 if dx > 0.8 else dx
dx = -0.8 if dx < -0.8 else dx
```

Detect the Placement Line

The robot is programmed to recognize specific placement lines corresponding to different block colours. Each block colour has a predefined number of placement lines. For example:

- **Red block**: Place on the first line (single line detected).
- **Green block**: Place on the second line (two lines detected).

```
position = {'red':1, 'green':2, 'blue':3, 'None':-1}
```

In the process of following line, the car will keep detecting the placement line. If the following condition is satisfied, which means the line is recognized.

```
if line_width > 100 and block_clamp:
```

Then determine the position of placement line:

```
if transversae_num == position[target_color]:
    place en = True
```

The width of the line is obtained by the following function:

```
if detect_step == 'line':
    line_center_x = center_x
    line_center_y = center_y
    line_width = data
```

After all the lines are recognized completely, the recognition function will be stopped to prevent the interference from repeat recognition of the same placement line.

```
if (time.time()-transversae_time) > 1:
    transversae_num += 1
    print(transversae_num)
    transversae_time = time.time()
```

Place the Block

When the number of recognized placement lines is equivalent to the numbers of placement lines corresponding to the placement position of the target block, the car will stop in the corresponding position and the robotic arm will be controlled to place the block to the corresponding position.

```
elif place_en:
    if time.time() >= place_delay:
        rospy.sleep(0.1)
        set_velocity.publish(0, 0, 0)
        target = ik.setPitchRanges((-0.24, 0.00, -0.04), -180, -180, 0)
        if target:
            servo_data = target[1]
            bus_servo_control.set_servos(joints_pub, 1200, ((6, servo_data['servo6']),))
            rospy.sleep(1)
            bus_servo_control.set_servos(joints_pub, 1500, ((3, servo_data['servo3']), (4, servo_data['servo4']),
            (5, servo_data['servo5'])))
        rospy.sleep(1.8)

        bus_servo_control.set_servos(joints_pub, 500, ((1, 150),))
        rospy.sleep(0.8)
```

Due to the limitations of the camera's detection range, the robot may lose sight of the line before reaching the target position. To address this issue, the program includes a delay mechanism that allows the robot to continue moving even when the line is temporarily out of view.

```
if transversae_num == 1:
    place_delay = time.time() + 1.1
elif transversae_num == 2:
    place_delay = time.time() + 1.1
elif transversae_num == 3:
    place_delay = time.time() + 1.2
```

END OF EXPERIMENT