KIE2002-LAB 4 SEM 1 2026/2027

EXPERIMENT KW14: Line Following with ArmPi Pro

Related course: KIE3010 (Instrumentation)

OBJECTIVES:

1. To understand the hardware and software components of the ArmPi Pro, including the camera, motors, and robotic arm.
2. To set up and calibrate the camera for real-time line recognition and path-following.
3. To analyse the impact of velocity changes on the robot's performance.

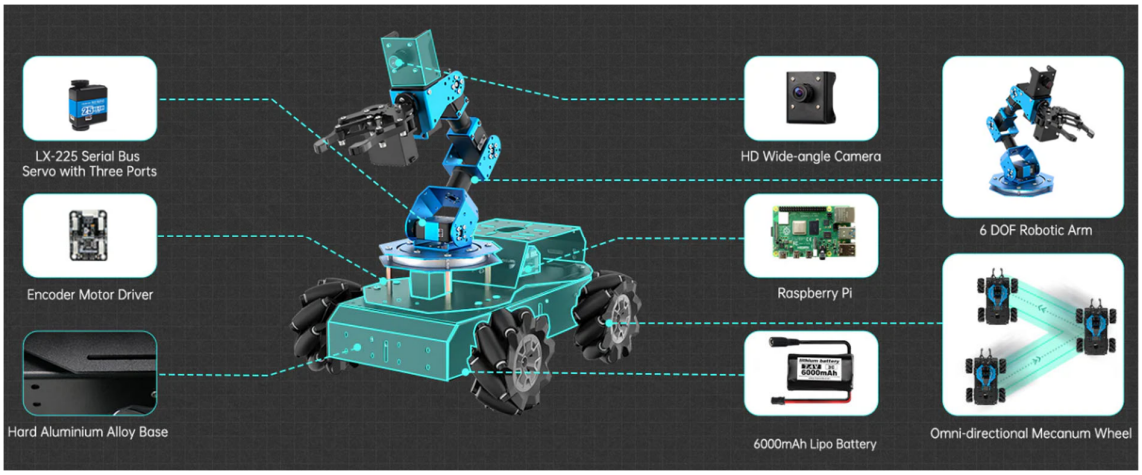
EQUIPMENT:

ArmPi Pro Robotic Kit, PC/Laptop with Wi-Fi connection, Tapes (3 colours), Measuring 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:



Powered by Raspberry Pi and based on ROS, ArmPi Pro is intelligent vision transporting robot and programmed in Python. On the basis of ArmPi FPV robotic arm, it is added an omni-directional mecanum wheel chassis, which robot can carry out mobile gripping, target tracking, intelligent transport and more functions.

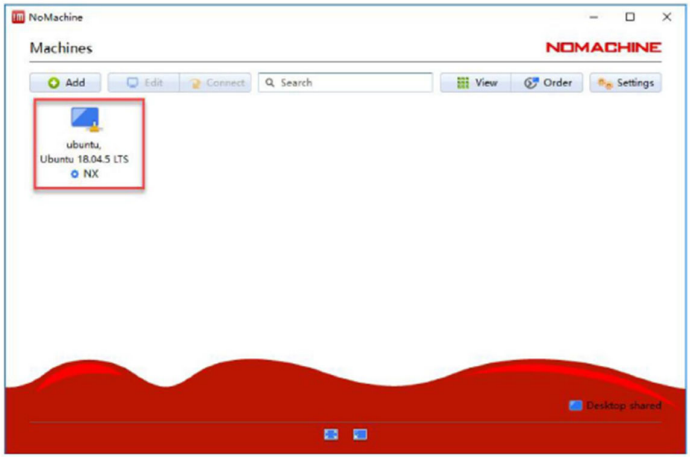
PROCEDURES:

A. Setup and Initialization

1. Switch on Raspberry Pi expansion board. The LED1 and LED2 of the Raspberry Pi will be on first and then the LED2 will flash every 2 seconds, which means the robot is turned on successfully.
2. After turning on, Raspberry Pi will be in an AP direct connection mode by default and launch a Wi-Fi hotspot starting with “HW”. Password : hiwonder
3. After connecting the hotspot, double click to open “NoMachine”.



4. Then double click ubuntu icon in the interface. At this time, a prompt interface will pop up, type Username and Password. The username is "**ubuntu**" and the password is "**hiwonder**", check "Save this password in the connection file" and click "Login". Then click "OK" in the following pop-up windows and you can find Raspberry Pi remote desktop.



B. Line Following

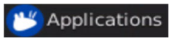
Recognize the colour and process it with LAB colour space. Firstly, convert RGB colour space to LAB and then perform binarization, dilation and erosion and other operations to obtain the outline of the target colour. Then frame the line and its centre point. Finally, after identifying colour line, ArmPi Pro will follow the line.

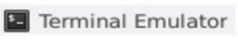
The source code of program is located in:

/home/ubuntu/armpi_pro/src/visual_patrol/scripts/visual_patrol_node.py

1. Place a red-coloured tape on a flat surface to create a line for the robot to follow. Ensure the line is at least 1 meter long for accurate measurement.
2. Use a measuring tape or ruler to measure the total length of the line (L) in meters. Record this value.
3. Turn on the ArmPi Pro and connect to the system desktop via No Machine.
4. Locate and open the source code of the program. Before starting the experiment, replace the existing code with the contents of the file located at:

   /home/ubuntu/Desktop/Code/visual_patrol_node.py

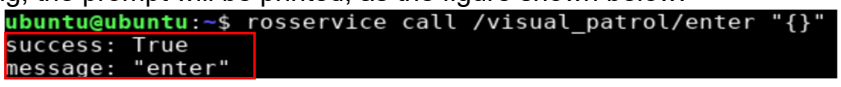
   Note: You may need to restart the robot following steps in Part D for the updated code to be executed.
5. Click 

in the lower left corner and select 

to enter the terminal. (You can also press "Ctrl+T" to open the terminal).
6. Open the terminal and enter the command to start the line-following game.

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

   After entering, the prompt will be printed, as the figure shown below:



7. 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.



8. Now, enter the terminal and run the game by entering the command:
```
rosservice call /visual_patrol/set_running "data: true"
```
9. Set the target line colour to red using the command:
```
rosservice call /visual_patrol/set_target "data: 'red'"
```
10. Use a stopwatch or timer to measure the time taken (t) for the robot to complete the line-following task.
11. Calculate the speed of the robot. Repeat the task at least 3 times to ensure accurate measurements. Record the time taken for each trial and calculate the average speed.
12. Repeat line-following task (steps 1 – 10) for two additional colours of tapes. Refer to Appendix 1 for steps to add a new recognition colour.

C. Varying the Velocity of the Motor

1. The motor control is handled in the visual_patrol_node.py file, specifically in the section where the `set_velocity.publish()` function is called. You can adjust the velocity parameters to control the robot's speed.
2. The `set_velocity.publish()` function takes three parameters:
   - **Linear Velocity** (first parameter): Controls the forward/backward speed of the robot.
   - **Direction Angle** (second parameter): Controls the direction of movement.
   - **Yaw Rate** (third parameter): Controls the turning speed.
3. Declare a global variable in the code and modify the linear velocity to 50 and 150, respectively.
4. After modifying the code, save the file and repeat the line-following task in Part B with the new velocity settings to collect speed data for three different colours of tapes.

> ***Important Note:*** *To ensure that your updated code runs correctly on the ArmPi Pro robot, follow the steps in Part D to safely shut down and restart the robot after making*

changes to the code. This prevents conflicts with running processes and ensures the latest version of your code is executed.

Stop and Exit the Game

1. Stop the game using the command:
```
rosservice call /visual_patrol/set_running "data: false"
```

2. Exit the game using the command:
```
rosservice call /visual_patrol/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.

DISCUSSION

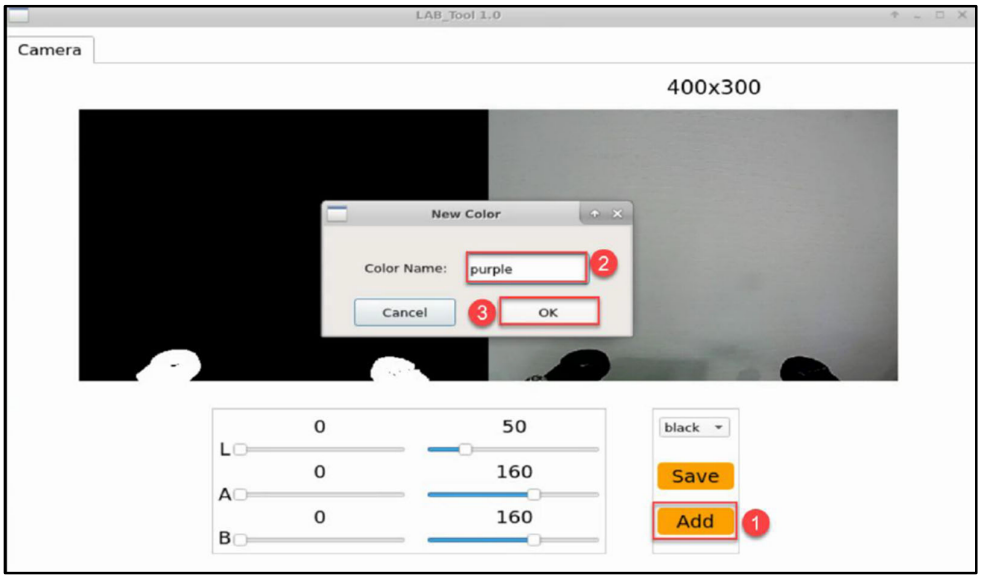
Compare the robot's performance at different speeds and analyse how velocity affects its ability to follow lines of various colours.

1. Compare the set velocity and measured speed to analyse how closely the robot's actual speed matches the programmed velocity. If there is an error, explain the possible causes.
2. Analyse how different velocity setting affects the robot's ability to follow lines of different colours. Does the robot perform better or worse at higher speeds?
3. Discuss whether the robot's ability to detect and follow lines of different colours is affected by the increased velocity.

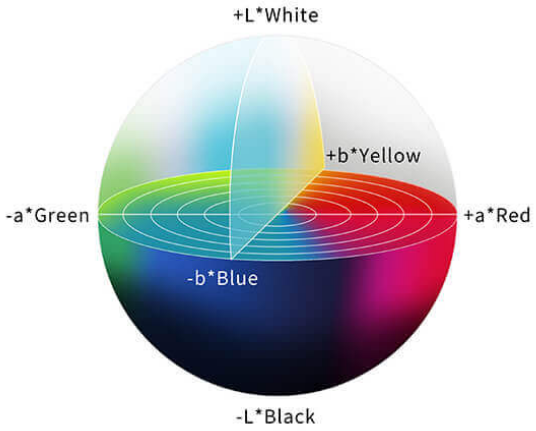
Appendix 1: Steps to Add a New Recognition Colour for Line Following

1. Enter the following command to open the colour threshold adjustment tool:
```
python3 lab_config/main.py
```

2. Once the tool opens, click the "Add" button in the lower right corner of the interface. Enter a name for the new colour (e.g., "purple") and confirm.

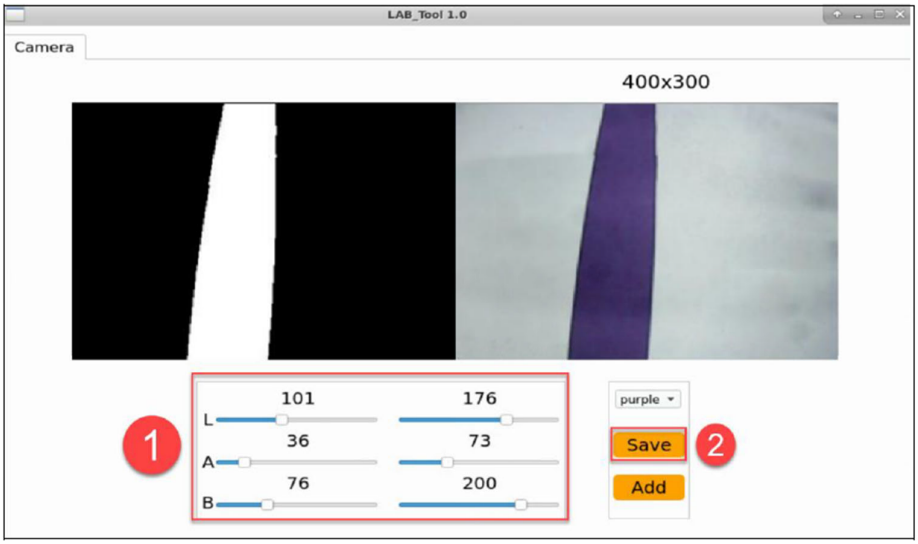


3. On the right side of the interface, you will see a real-time image from the camera. Point the camera at the coloured tape (e.g., purple) that you want the robot to recognize.
4. Use the six slider bars to adjust the LAB colour thresholds:
    a. **L (Lightness):** Adjust the lightness range.
    b. **a (Green to Red):** Adjust the green-to-red colour range.
    c. **b (Blue to Yellow):** Adjust the blue-to-yellow colour range.



Note: You may use Adobe Photoshop to obtain the LAB values.

Adjust the sliders until the target colour area (e.g., purple) appears **white** in the processed image, while other areas appear **black**.



APPENDIX 2: IMAGE PROCESSING FOR LINE FOLLOWING

The ArmPi Pro utilizes image processing techniques to detect and follow a designated coloured line. The key steps involved in this process include:

- Converting images from RGB to LAB colour space.
- Applying binarization, dilation, and erosion operations.
- Extracting contours and identifying the centre of the detected line.
- Controlling motor movements based on image analysis.

Colour Space Conversion

To enhance colour recognition, the RGB image is converted to LAB colour space, which separates luminance (L) from colour information (A and B). This allows for more effective colour thresholding, reducing the impact of lighting variations.

Binarization

Binarization is used to create a black-and-white image where the desired colour is white and everything else is black. The `inRange()` function in the cv2 library is used to binarize the image:

```
frame_mask = cv2.inRange(frame_lab, tuple(color_range['min']),
                         tuple(color_range['max']))
```

- `frame_lab`: Input image in LAB color space.
- `color_range['min']`: Lower threshold for colour.
- `color_range['max']`: Upper threshold for colour.

Morphological Operations (Dilation and Erosion)

Morphological operations refine the binary image by reducing noise (erosion) and improving the clarity of detected lines (dilation).

```
eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))
dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))
```

Contour Detection

After processing the above image, obtain the contour of the recognition target. The `findContours()` function in cv2 library is involved in this process.

```
contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

Find the maximum contour from the obtained contours. To avoid interference, set a minimum value. Only when the area is greater than this minimum value, the target contour will take effect. The minimum value here is “50”.

```
area_max_contour, area_max = getAreaMaxContour(contours)  # find the biggest contour
if area_max > 50:  # there is the biggest contour
```

Position Calculation

The `minAreaRect()` function in cv2 library is used to obtain the minimum external rectangle of the target contour, and the coordinates of its four vertices are obtained through the `boxPoints()` function. Then, the coordinates of the centre point of the rectangle can be calculated from the coordinates of the vertexes of the rectangle.

```
rect = cv2.minAreaRect(area_max_contour)  # The minimum enclosing rectangle
box = np.int0(cv2.boxPoints(rect))       # The four vertices of the smallest enclosing rectangle
for i in range(4):
    box[i, 1] = box[i, 1] + (n - 1)*roi_h + roi[0][0]
    box[i][0] = int(Misc.map(box[i][0], 0, size_s[0], 0, img_w))
    box[i][1] = int(Misc.map(box[i][1], 0, size_s[1], 0, img_h))
cv2.drawContours(img, [box], -1, (0, 255, 255), 2)  #draw a rectangle composed by four points
#Get the opposite corners of a rectangle
pt1_x, pt1_y = box[0, 0], box[0, 1]
pt2_x, pt2_y = box[1, 0], box[1, 1]
pt3_x, pt3_y = box[2, 0], box[2, 1]
center_x = int((pt1_x + pt3_x) / 2) #center point
center_y = int((pt1_y + pt3_y) / 2)
line_width = int(abs(pt1_x - pt2_x))
cv2.circle(img, (center_x, center_y), 5, (0,0,0), -1) # draw center point
```

END OF EXPERIMENT