Introduction

The project serves as an example of using robotics to solve real life problems and integrating them to applications to increase productivity. The project demonstrates this through the practical implementation of a calculator operating robotic arm system achieved by using the Braccio robotic arm. The program accepts 2 integer inputs through the serial monitor and does basic arithmetic calculations. Furthermore, it displays the results of the operations visually by moving the arm to the appropriate locations.

Requirement analysis

The robotic arm calculator project has specific functional and non-functional requirements to ensure its successful operation. These requirements are divided into two categories: functional, which focus on the robot's capabilities and user interaction, and non-functional, which address performance, safety, and usability considerations.

A. Functional requirements

The robotic arm has 6 degrees of freedom that allows it to make movements across 6 different axes. This degree of freedom allows the arm to accurately press buttons of the calculator. The calculator is set up in a fixed position on the ground with a 5x3 grid layout. The arm then moves to those positions based on the input received. The arm moves to those locations by moving its 6 independent motors at the same time and directs its gripper onto the centre of the box square.

To run the robot arm it requires input from the serial monitor and the input needs to be of maximum two digit integers. The program has input validation that ensures that the input is within the constraint otherwise it would reject that value. Besides two integer inputs it also takes an operator (+, -, *, /) as a character and based on the operator chosen it does the mathematical calculations. After receiving the input it then parses it to ensure that the input fits the constraint before sending it off to another function responsible for movement.

Initially to find the coordinates of all the 15 positions, extensive trial and error was made by testing out using different values for the motors. At first the first 3 coordinates of the first row were found and were then stored onto the code, after setting up those coordinates all the other coordinates were fairly easy to find since the initial 3 coordinates just needed to be adjusted to fit the needs of the current ones. After receiving the parsed input the arm then proceeds to press the button corresponding to the digit entered. The arm initially moves its M1 motor and positions itself towards the digit's location; later the arm moves its M2, M3 and M4 in tandem but also ensuring that the movement is well within the safe values of the motor in order not to damage the motors. The arm then slowly descends the M3 and M4 motor so that the grippers at the end can mimic the finger of a human arm pressing a button. The general movement of the robot is similar with some distinct differences due to the fact how the digits are layout on the

grid. In addition, after completing the designated sequence of movements it returns back to the default position signalling the operation's completion.

All the mathematical calculations that are being done are carried out in a different function separated from the main movement code. This is done to ensure there is modularity between the code blocks and making sure each part of the code only does those necessary parts. Moreover, the calculation function also has error handling to ensure there is no zero division, if there is one the program returns 1 indicating an error.

B. Non-functional requirements

To ensure that the robot can perform its tasks safely, many safety considerations were taken into account. In order not to put too much stress on the motors of the robot delays were added between each sequence. This delay allows the motor to cool down a little before going on to the next task. Furthermore, when the robot is pressing a button there is a considerable distance between the grippers of the robot and the grid. This is mainly done so that the grippers are not damaged when the robot is pressing a button.

Besides safety considerations there are some limitations of this implementation of the program. The robot was placed in an isolated environment which is good since there are less external factors affecting the function of the robot but there is a caveat. The programming was completely done remotely without any interaction with the robot solely relying on a video feed. This resulted in there being parallax error during button presses and also the video feed made it difficult to see whether the arm was even pressing the right button since the two views of the camera were not enough and led to a lot of assumption. These assumptions and errors led to some reliability of the system which could potentially be resolved if work was done on a physical environment.

Code explanation

The code overall is modularised consisting of multiple functions which are responsible for specific tasks. The code firstly consists of 'setup()' and 'loop()' functions which initialises and takes user input. The `performCalculation()` function executes specified arithmetic operations, while `getValidInput()` ensures inputs are numeric and within two digits. Additionally, `getOperationInput()` collects a valid operator. Movement control is managed by the `operation()` function, which maps user inputs to predefined robot movements. Input processing involves reading from the serial monitor, validating, and handling inputs sequentially. The code includes safeguards against division by zero and checks for valid input size and operation type, enhancing safety. Error handling is implemented through prompts that guide users for valid inputs and alert them to issues, such as division by zero, ensuring a smooth interaction with the robot.

Results

The system effectively processes two-digit integer inputs and performs calculations through the robotic arm's movements. Testing revealed consistent performance in basic arithmetic operations (+, -, *, /), with the arm accurately pressing calculator buttons according to input sequences.

Despite the challenges of remote programming and parallax errors from video feed limitations, the system maintains functional reliability. The implemented safety measures, including motor cooling delays and controlled gripper movements, proved effective in preventing hardware damage. Error handling successfully manages invalid inputs and edge cases like division by zero.

Future improvements could focus on enhancing precision in button pressing and implementing a more sophisticated visual feedback system. The current implementation serves as a practical demonstration of robotic automation, successfully integrating user input, mechanical movement, and mathematical operations.