Robotics 2

ARI3215

Assignment Report

Amy Spiteri, Clyde Vella, Daniel Pace

January 29, 2026

Contents

1 Introduction

In this assignment, we were tasked with designing, implementing, and demonstrating an autonomous robot or simulation that can sense, reason, and act autonomously, demonstrating key aspects of embodied Artificial Intelligence (AI). A hardware implementation using the Elegoo Smart Robot Car V4 was chosen. Our robot is a gesture-controlled robot that responds to hand gestures in real-time, captured by a webcam. The system uses computer vision techniques to recognize specific gestures made by the user and translates them into movement commands for the robot. The robot also incorporates an ultrasonic sensor to detect nearby obstacles and prevent collisions and a gyroscope to maintain accuracy with turning.

All of the project code can be found on the following GitHub repository: `https://github.com/spiteriamy/ari3215-robotics2`. The final working python code can be found in the subdirectory `python/`. The final working Arduino code can be found in the subdirectory `arduino/robot/`.

2 System Design

The system was designed as a two-layer architecture that separates high-level perception and decision-making code from low-level real-time motor and sensor control. The Raspberry Pi hand in hand with the camera performs computationally heavier computer vision tasks (webcam capture, hand landmark detection, and gesture classification), while the Arduino remains responsible for deterministic hardware interaction (motor driving, ultrasonic ranging, gyroscope-assisted turns, and feedback devices). This split improves reliability and responsiveness: the Raspberry Pi can interpret gestures at a controlled frame rate, and the Arduino can execute movement and safety behaviour without being blocked by vision processing.

2.1 High-Level Architecture

Figure 1 illustrates the end-to-end control loop. The workflow begins with the Raspberry Pi acquiring a real-time video stream from the webcam. Frames are processed using computer-vision algorithms to detect hands, extract landmarks, and classify the current gesture. If no valid gesture is detected, the system continues to sample frames until a command is recognised.

Once a gesture is recognised, the system identifies the intended action and maps it to a compact robot command. This command is transmitted to the Arduino over USB serial. The Arduino then drives the motors and performs the associated sensing checks (ultrasonic and gyroscope) during execution. The loop continues until either a new command is detected (and transmitted) or the session ends.

2.2 Gesture-to-Command Design

Gesture control was designed to be intuitive and parameterised, allowing the user to specify both direction and movement magnitude. Directional intent is provided through distinct gestures (e.g., forward/backward/left/right/stop), while separate gestures provide parameters such as duration for linear movement and angle for turning. This avoids requiring many unique gestures while still supporting multiple speeds/durations/turn angles.

A dedicated stop gesture acts as an immediate safety and usability feature, enabling the user to halt the robot at any time. A separate “secret” gesture was also implemented as a special command used for demonstration/feedback purposes.

2.3 Subsystem Responsibilities

Raspberry Pi (Perception and Decision Layer).

The Raspberry Pi is responsible for:

- Capturing the webcam feed and running hand detection/tracking.
- Classifying gestures and converting them into high-level robot intentions.

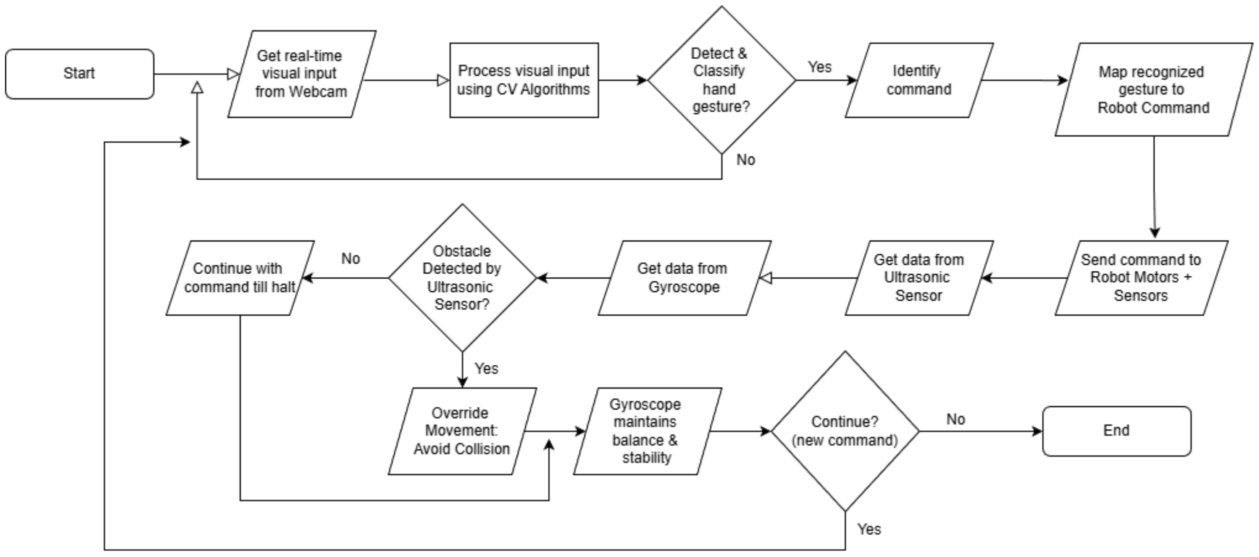
Start
Get real-time visual input from Webcam
Process visual input using CV Algorithms
Detect & Classify hand gesture?
Yes
No
Identify command
Map recognized gesture to Robot Command
Send command to Robot Motors + Sensors
Get data from Ultrasonic Sensor
Get data from Gyroscope
Obstacle Detected by Ultrasonic Sensor?
No
Continue with command till halt
Yes
Override Movement: Avoid Collision
Gyroscope maintains balance & stability
Continue? (new command)
No
End
Yes

Figure 1: Flowchart of System

- Encoding intentions into a concise serial command format.
- Sending commands only when a change is detected, reducing redundant serial traffic and improving stability.

Arduino (Actuation and Safety Layer). The Arduino is responsible for:

- Decoding received serial commands and triggering motor actions (forward, backward, left, right, stop).
- Reading ultrasonic sensors to detect obstacles and enforce collision avoidance behaviour.
- Using gyroscope readings to improve turning accuracy and reduce drift during rotation.
- Driving feedback hardware (e.g., LED strip / buzzer) to indicate system state and received commands.

2.4 Safety and Autonomous Overrides

Although the primary interaction is user-driven gesture control, the robot exhibits autonomous behaviour through safety overrides. During motion, ultrasonic distance checks provide collision prevention: if an obstacle is detected within a defined threshold, the Arduino can inhibit or override movement to avoid contact. This design ensures that even if a gesture is misclassified or the user issues a command unsafely, the robot still prioritises physical safety.

In addition, gyroscope-assisted turning improves repeatability of angular movements by using orientation feedback to correct for real-world variance in motor response, battery level, and surface friction. This stabilisation mechanism supports more consistent behaviour during repeated trials and demonstrations.

2.5 Software Organisation

The repository is organised into two main codebases:

- `python/`: Raspberry Pi scripts for camera input, gesture recognition, decoding, and serial transmission (e.g., the main entry point and the gesture-to-command mapping module).
- `arduino/robot/`: Arduino firmware that receives commands and controls motors and sensors, supported by a movement control library used to abstract repeated drive/turn logic.

This separation mirrors the architectural split between the perception layer (Raspberry Pi) and the actuation/safety layer (Arduino), making the system easier to test, debug, and extend independently.

3 Implementation

To implement the system described above, several components were integrated, including hardware setup, computer vision for gesture recognition, sensor fusion for autonomous behaviour, communication protocols, and control algorithms.

3.1 Hardware Integration & Setup

The hardware setup involved configuring the Elegoo Smart Robot Car V4 with an ultrasonic sensor for obstacle detection and a gyroscope for orientation tracking. The robot's motors were connected to the Arduino board, which served as the central controller. The ultrasonic sensor was wired to the appropriate pins on the Arduino to enable distance measurement, while the gyroscope was set up to provide real-time orientation data. Serial communication between the Arduino and a Raspberry Pi was established to facilitate command transmission based on gesture recognition.

Aside from the default components of the Elegoo Smart Robot Car V4, an additional ultrasonic sensor was added to extend obstacle detection to rear obstacles. A buzzer was also integrated to provide auditory feedback, and a LED strip was included for visual feedback based on the robot's actions.

On the side of the Raspberry Pi, a webcam was connected to capture real-time video feed for gesture recognition. The Raspberry Pi ran a Python script that processed the video input, recognized hand gestures, and sent corresponding movement commands to the Arduino via serial communication. The Raspberry Pi was configured to automatically run the gesture recognition script on startup, ensuring that the system was ready for operation as soon as it was powered on.

3.2 Computer Vision Pipeline

The computer vision pipeline was implemented using the MediaPipe library for hand detection and tracking [1]. A webcam connected to the Raspberry Pi captured real-time video feed, which was processed to identify hand gestures. The MediaPipe Hands solution provided 21 3D landmarks for each detected hand [1], which were used to recognize specific gestures such as open hand, closed fist, and pointing. Custom algorithms were developed to interpret these landmarks and classify them into predefined gestures. The recognized gestures were then translated into movement commands for the robot.

An experimental alternate method of decoding hand gestures, where the angle of the pointing finger is read, and the robot is directed accordingly, was eventually abandoned due to instability and inaccuracy in gesture recognition.

3.3 Sensor Fusion & Autonomous Behaviour

The robot's autonomous behaviour was achieved by fusing data from the ultrasonic sensors, gyroscope, and gesture recognition system. The ultrasonic sensor monitored the distance to nearby obstacles only when necessary and not continuously, allowing the robot to make real-time decisions to avoid collisions. When an obstacle was detected within a predefined threshold, the robot would halt or change direction based on the last received gesture command. The gyroscope provided orientation data, which was used to enhance the robot's turning accuracy. When a turn command was issued via gesture recognition, the gyroscope data ensured that the robot executed the turn precisely, compensating for any drift or inaccuracies in movement.

3.4 Communication Protocol

The communication between the Raspberry Pi and the Arduino was established using serial communication over USB. The Raspberry Pi sent encoded movement commands to the Arduino based on the recognized gestures. A simple protocol was designed where each command corresponded to a specific action, such as moving forward, backward, turning left, or turning right, and by how much. The Arduino decoded these commands and executed the corresponding motor actions. Feedback from the ultrasonic sensor and gyroscope was also sent back to the Raspberry Pi when necessary to inform decision-making processes. The serial outputs from the Arduino were read and printed to the console on the Raspberry Pi for real-time monitoring and debugging purposes.

3.5 Control Algorithms

Control of the movements was handled through a custom intermediary library built specifically for the Elegoo Smart Robot Car V4 for our purposes. The main functions included in this library were to move forwards, backwards, turn left, turn right, and stop. Each function took parameters to specify speed and duration where applicable. The obstacle avoidance was handled inside the main control loop, where the ultrasonic sensor readings were checked using custom functions that limited the wait duration for a reading to avoid blocking the main loop. If an obstacle was detected within a certain distance, the robot would stop and wait for a new gesture command before proceeding. The gyroscope data was used to ensure accurate turning by adjusting the turn duration based on the current orientation of the robot.

3.6 Efficiency & Robustness

To ensure efficient hand detection and communication between the Raspberry Pi and Arduino, several optimizations were implemented. The gesture recognition algorithm is invoked at a controlled frame rate to balance responsiveness and computational load. The serial communication protocol was designed to minimize latency by using concise command encoding, and a command is only sent when a new gesture is detected, reducing unnecessary data transmission.

Error handling mechanisms were incorporated to manage potential issues such as incorrect gesture execution, communication failures, and sensor malfunctions. The system includes checks to verify the integrity of received commands and sensor data, displaying error messages on the console, and displaying error codes via the LED strip on the robot.

4 Testing & Evaluation

Testing of the system was carried out continuously throughout the development process in order to identify issues, improve performance, refine the robot's behaviour, ensure correct integration of the various components, and address implementation errors as they arose. Once development was complete, additional testing was then conducted on the final system to assess the overall performance of the robot.

4.1 Testing Methodology

Testing was conducted iteratively throughout development, beginning with component-level testing leading to a full system evaluation at the end. Initial experimentation with the MediaPipe library was carried out to verify reliable hand detection, finger tracking, and the ability to recognise the number of fingers shown, forming the basis for the gesture-based control. Initial testing also focused on validating serial communication over USB between the Arduino and Raspberry Pi by transmitting test messages and confirming they were received and decoded correctly.

Once basic functionality had been confirmed, physical testing of the robot prototype was performed in a variety of environments. The majority of testing took place indoors under varying lighting conditions, including both natural daylight and artificial indoor lighting. The robot was mostly tested on smooth tiled flooring. Limited outdoor testing was also conducted under direct sunlight on a rougher concrete surface to observe the effect of environmental changes on both gesture recognition and robot movement. During testing, the distance between the user's hand and the camera was typically approximately one metre, as the robot was placed on the floor with the camera facing upwards while the user stood above it.

Each gesture command was tested repeatedly using different duration and turning angle parameters. Sequences of different commands were also tested to evaluate how the robot handled successive movement instructions and transitions between different actions. These tests were repeated across the different environments and lighting conditions described above to assess consistency and robustness of the robot.

Obstacle detection testing was performed by placing objects in front of or behind the robot and measuring the distances that the robot stopped at from them using a ruler. This allowed verification that the robot stopped at approximately the predefined threshold distance when an obstacle was detected. Additionally, to prevent the robot from turning when an obstacle was found to the left or right of the robot, checking for obstacles to the side was also performed using the ultrasonic sensor before turning took place. This was tested by placing objects to the side of the robot. Turning accuracy was evaluated visually by observing whether the robot rotated by approximately the commanded angle, supported by the gyroscope assisted correction.

To help with testing and debugging, an LED strip was added to the robot to provide visual feedback on the commands received by the Arduino. After a hand gesture was detected and translated into a movement command, the LED strip displayed a colour corresponding to the type of command received. The strip consisted of five LEDs, with the number of illuminated LEDs representing the duration or angle associated with the command. This visual feedback made it easier to determine whether gestures were being correctly recognised and whether the intended commands were being transmitted and executed, allowing issues in gesture classification or command interpretation to be identified efficiently.

4.2 Gesture Recognition Evaluation

The gesture recognition component of the system was tested through repeated execution of the different gesture commands. Evaluation focused on the correctness, consistency, and responsiveness of gesture detection, as well as the system's robustness to variations in lighting, hand positioning, and user movement. In general, gestures with clear visual distinctions, such as an open hand for the secret command and a closed fist used for stopping, were recognised reliably across most testing conditions. These gestures showed consistent performance in indoor environments under both natural and artificial lighting, with minimal misclassification.

Gestures involving direction, such as turning left or right and moving forwards or backwards, were more sensitive to variations in hand orientation and camera perspective, and sometimes confused with each other. In particular, certain finger positions or rapid hand movement occasionally resulted in incorrect classification. Despite this, repeated gesture attempts generally resulted in correct recognition, and misclassifications were typically resolved by presenting the gesture again.

The responsiveness of the gesture recognition system was observed to be sufficient for real-time control. The delay between presenting a gesture and the robot initiating the corresponding movement was short enough to allow intuitive interaction. The decision to only transmit commands when a change in gesture was detected further reduced unnecessary communication and contributed to more stable system behaviour.

Overall, the evaluation demonstrated that the gesture recognition system was mostly effective for controlling the robot. However, performance remained dependent on clear user gestures, showing some limitations.

4.3 Robot Motion Accuracy

The accuracy of the robot's motion was evaluated by observing how closely the executed movements matched the commands given. Evaluation focused on forward and backward motion consistency and turning accuracy.

Movement commands were tested at different duration settings. The robot was observed to move in a generally consistent manner for repeated executions of the same command on smoother surfaces, with only minor variations in travelled distance. Additionally, visual inspection showed that the robot was able to approximate the intended turning angles with reasonable accuracy and consistency. Any turning errors were typically within a small difference in angle.

Overall, the motion control system provided sufficient accuracy for gesture-based interaction. While perfectly precise positioning was not achievable due to hardware and environmental limitations, the level of accuracy achieved was appropriate for controlling the robot's movement.

4.4 Obstacle Detection & Safety Testing

Obstacle detection was evaluated by observing the robot's behaviour when objects were placed at varying distances along its path of motion, as well as its behaviour when obstacles were placed to the left/right of the robot. Testing focused on the reliability of obstacle detection, the consistency of stopping and turn prevention behaviour, and the system's ability to prevent collisions during gesture-controlled operation.

During testing, obstacles were positioned at known distances in front of, behind, and to the sides of the robot, and these distances were measured using a ruler. The ultrasonic sensors were observed to detect obstacles at approximately the predefined threshold distance, causing the robot to stop/prevent motion as intended.

The robot was also tested under scenarios where obstacles were introduced while the robot was already in motion. In these cases, the robot was able to respond in a timely manner by halting movement before making contact with the obstacle. This contributed to safer operation and reduced the risk of unintended collisions during gesture-controlled use. Rear obstacle detection provided similar behaviour when reverse movement commands were given.

Certain limitations of ultrasonic sensing were observed during testing. Detection reliability was influenced by the position of nearby objects. Obstacles positioned at extreme angles relative to the sensor were more difficult to detect reliably and often went unnoticed, leading to collisions.

4.5 System Robustness & Reliability

The overall robustness of the system was evaluated by observing its behaviour during extended use. Testing focused on the stability of communication between components, the system's ability to handle temporary errors or misclassifications, and the consistency of behaviour over time.

During prolonged operation, the serial communication link between the Raspberry Pi and Arduino remained stable, with commands being transmitted and executed without noticeable interruption. The decision to only send commands when a change in gesture was detected helped to reduce unnecessary communication and contributed to reliable real-time performance. No persistent communication failures were observed during typical testing sessions.

Although gesture misclassification was present in some situations, it did not generally lead to unsafe or unpredictable robot behaviour. Because commands were short in duration, incorrect movements were easily corrected by issuing a new gesture. The use of obstacle detection further reduced the risk associated with occasional recognition errors, as the robot would halt if an obstacle was detected within the threshold.

4.6 Discussion of Results

The testing and evaluation process demonstrated that the system was capable of providing intuitive real-time control of the robot through vision-based hand gestures, while maintaining a basic level of autonomous safety through sensor integration. The combination of gesture recognition, motion control, and obstacle detection allowed the robot to operate in a way that balanced user input with environmental awareness.

Gesture recognition performed reliably for clearly distinguishable gestures, particularly those involving open or closed hands. Performance was more varied for gestures that depended on different finger positioning, which shows a limitation of vision-based systems. The decision to use a small set of simple and discrete gestures was therefore effective, as it prioritised robustness over more expressive control.

The motion control system demonstrated predictable behaviour, with gyroscope-assisted turning improving the consistency of turning. Although minor inaccuracies in distance travelled and turning angle were observed, these did not significantly affect the overall usability.

Obstacle detection was important in improving operational safety. The ultrasonic sensors successfully prevented most direct collisions and acted as a safeguard against occasional gesture misclassification. However, testing also highlighted the limitations of ultrasonic sensing. This shows the trade-off between simplicity and sensing capability in robot systems.

The evaluation shows that the project integrates perception, decision-making, and physical action into a cohesive embodied AI system. While performance was influenced by hardware constraints and environmental factors, the design choices made throughout development achieved a practical balance between complexity, reliability, and real-time responsiveness.

4.7 Limitations & Future Improvements

Although the system achieved reliable real-time gesture-based control, several limitations were identified during testing. Many of these arise from hardware constraints, vision-based sensing, and the scope of the project.

A primary limitation is the dependence on lighting conditions and clear hand visibility for gesture recognition. Performance decreased in poor lighting, when the hand was partially not visible, or when gestures were performed with certain hand angles. These challenges are characteristics of vision-based systems and could be mitigated in future work through the use of more advanced camera systems, machine learning models trained on a wider range of hand appearances and environmental conditions, or a more advanced method of classifying hand gestures.

The gesture set itself was intentionally limited to a small number of commands to prioritise reliability. While this improved robustness, it also restricted the amount and flexibility of control. Future improvements could explore adaptive gesture sets, or hybrid control methods that combine gestures with additional input types, such as voice commands.

While obstacle detection was effective for basic collision avoidance, it was limited by the characteristics of ultrasonic sensing. Detection reliability varied depending on the object's angle relative to the sensor. The system was also reactive rather than predictive, stopping only once an obstacle was within a predefined threshold rather than planning alternative paths. Future improvements could include additional sensors (such as infrared or LiDAR) and the implementation of path-planning or mapping algorithms to enable more intelligent navigation.

Finally, the evaluation process itself represents a limitation. Although the system was extensively tested across many scenarios, the evaluation focused mainly on behavioural correctness and robustness, and formal quantitative metrics such as precise accuracy percentages and latency measurements were not recorded during development. Future work should incorporate automated logging and structured performance measurement to allow for a more rigorous evaluation.

Overall, these limitations highlight opportunities for extending the system beyond a proof-of-concept towards a more robust and autonomous gesture-controlled robot.

5 Ethical Considerations

The MediaPipe library was used for hand detection and tracking. Key ethical issues relevant to the use of MediaPipe and similar computer vision libraries include data privacy, surveillance, bias, and discrimination. Because the system relies on a live webcam feed, privacy and surveillance are concerns that should be considered. Individuals in the background may be captured unintentionally without their knowledge or consent. To reduce privacy risks, users should be clearly informed when the camera is active (for example through the LED indicator on the webcam used). Additionally, informed consent should be obtained before recording or testing around other people. Privacy concerns can also be mitigated by avoiding the collection of unnecessary data, such as performing all video processing locally and avoiding storage of footage. If any footage must be stored, users should be informed about retention time, storage location, who has access, and the intended use of the data. The system relies only on a live video feed, and all processing of the video is done locally in real-time, and therefore no footage is permanently stored. Another important ethical concern is bias, fairness, and discrimination. Hand tracking accuracy can perform differently depending on a number of factors. This includes lighting conditions, skin tone, hand size, and anatomical differences. These performance differences may disadvantage users from underrepresented groups if tracking is less reliable for certain types of hands. Additionally, gesture-based interaction may also introduce accessibility issues. For example, users with limited hand mobility, missing fingers, or one-handed users may not be able to use the system effectively. Since the system controls a physical robot in the real world, physical safety is another ethical concern to consider. The use of robots in the real world can lead to robots colliding with objects, people, or pets. Misclassification of gestures can also lead to unintended movements. To mitigate these risks, safety mechanisms such as an emergency stop and safe speed limits were included in the robot.

References

[1] F. Zhang *et al.*, “Mediapipe hands: On-device real-time hand tracking,” 2020. [Online]. Available: https://arxiv.org/abs/2006.10214