Follow - me cart: A smart shopping assistant

By

Gu, Jiaming

Huang, Zixuan

Qiao, Shi

Final Report for ECE 445, Senior Design, Fall 2025

TA: Cui, Shengkun

10 December 2025

Project No. 03

Abstract

This project introduces a semi-autonomous Follow-Me Cart aimed at enhancing convenience and alleviating physical strain for shoppers. The system incorporates a Raspberry Pi for high-level perception and an ESP32 microcontroller for low-latency motor control. A lightweight YOLO-based detector combined with a Kalman-filter tracker enables human-following functionality by integrating camera bearings with ultrasonic radar distance to produce reliable motion targets. The cart employs two PID loops for real-time control of its speed and direction. Bluetooth commands enable the user to monitor, pause, and halt the cart in emergency situations. A power subsystem provides regulated outputs of 12 V, 5 V, and 3.3 V for sensing, computing, and actuation purposes. The experimental results indicate stable target tracking within the frequency range of 5–10 Hz and consistent following at a distance of 1–2 meters. The system demonstrates a minimum of 40 minutes of continuous operation, thereby confirming its efficacy as a smart, user-assistive shopping cart platform.

Contents

1 Introduction

1.1 Problem

In supermarkets, shopping centers, and malls, consumers frequently encounter difficulties maneuvering highly laden carts while concurrently examining products, using mobile devices, or supervising youngsters [1]. This circumstance diminishes both comfort and efficiency, resulting in a subpar shopping experience.

The increasing prevalence of smart retail and service robots has created a desire for semi-automatic shopping carts that alleviate physical strain, enabling customers to shop more comfortably and efficiently. Previous endeavors, including autonomous luggage and robotic carts, have established feasibility, underscoring the need to develop such a system.

A semi-automatic shopping cart system that autonomously follows clients is required, along with an intuitive mobile application for cart control, enhancing convenience, minimizing physical exertion, and elevating the shopping experience for diverse users.

1.2 Solution

To tackle this difficulty, we offer a Follow-Me Cart system operated through a remote controller. The architecture incorporates a Raspberry Pi [2] for high-level processing functions, including sensor fusion, user tracking, and app connection, while an ESP32 microcontroller manages low-latency motor control and ultrasonic safety functionalities. This hybrid design amalgamates these components and adeptly harmonizes computational intelligence with real-time dependability.

The Pi analyzes data from an ultrasonic sensor [5] and camera [6] to monitor the user, while the ESP32 [3] guarantees safety and facilitates low-latency motor reaction. This architecture will harmonize advanced intelligence with dependable real-time control.

The cart links to a remote controller via Wi-Fi or infrared rather than being fully autonomous. Customers can activate follow-me mode, regulate the vehicle's speed, modify the following distance, and halt or initiate the cart as required. The user possesses an infrared remote, whereas the shopping cart prototype is outfitted with a receiver and basic motor control. The cart can sustain a following distance of 1–2 meters in an unobstructed testing environment and can halt or circumvent impediments encountered. The navigation system may rely on a static map, as supermarket layouts are typically unchanging. The cart requires simply fundamental obstacle

avoidance rather than dynamic pathfinding when confronted with individuals or items. Provided the routes remain accessible, the cart can arrive at its destination. When a path is entirely obstructed, the cart may halt, which is permissible as it falls beyond the system's obligations. To ensure affordability and safety, we will develop the system at a reduced scale. The dimensions are reduced to accommodate around 50–70% of the weight of a standard shopping cart, with a payload capacity of 5–10 kilograms, in contrast to the typical capacity of 30–40 kg.

1.3 Block diagram

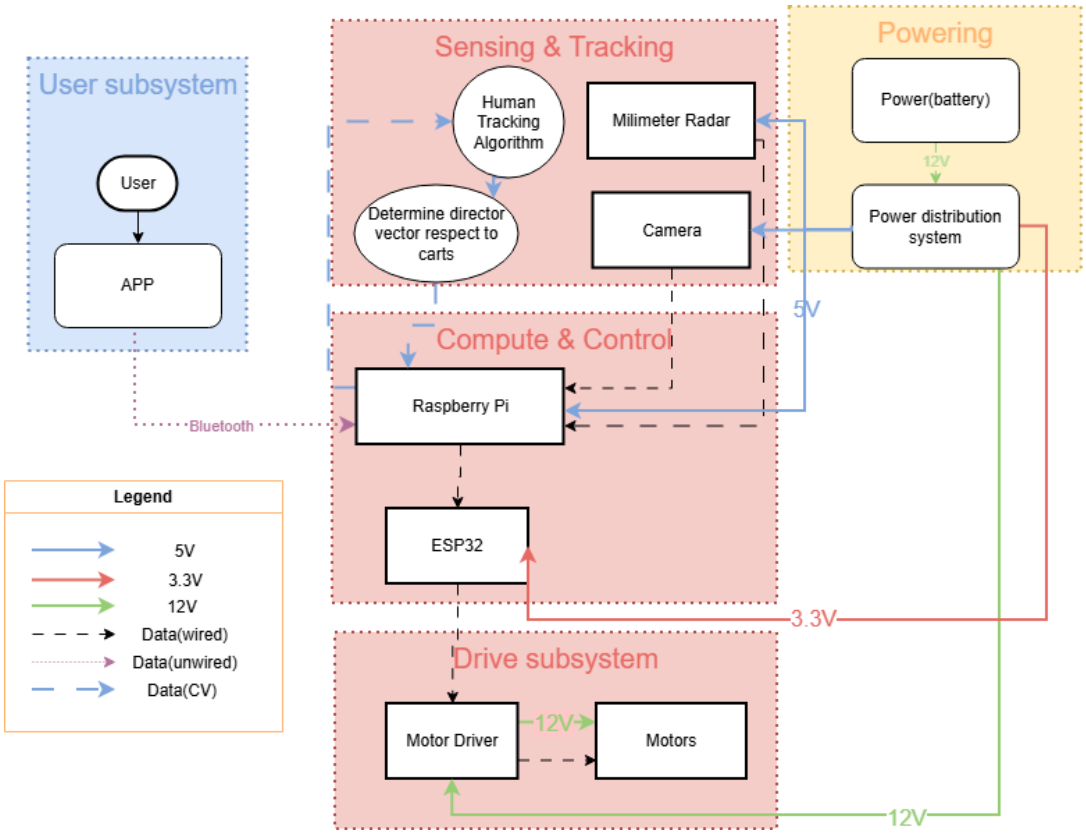

Figure 1 Block diagram of the Follow-me cart system

2 Design

2.1 Design procedure

2.1.1 Sensing and Tracking Subsystem

The Sensing and Tracking subsystem is designed to determine the user's distance from the cart and their proximity to the camera's center. Multiple approaches were evaluated, including

1. sensor-only tracking
2. vision-only tracking, utilizing bounding-box geometry, and
3. a hybrid method combining camera and radar.

The sensor offers reliable distance measurements but lacks angular resolution, whereas vision provides accurate angular information but yields unstable distance estimates. We selected a camera-based angular estimate in conjunction with a vision-based size cue for distance, which satisfies real-time constraints on the Raspberry Pi and generates the two signals necessary for closed-loop control by the ESP32.

A lightweight YOLO detector detects individuals in each frame and provides the bounding box coordinates. The horizontal pixel offset, denoted as err_x, where cx represents the detected box center and u0 signifies the midpoint of the camera frame. The Raspberry Pi calculates the normalized heading error, $err_{x,rad}$, utilizing the established horizontal field-of-view (HFOV). This metric indicates the user's deviation to the left or right.

The bounding-box height serves as an indicator of distance, exhibiting an inverse correlation with the user's physical proximity to the camera. A reference height h_0 is established at a nominal distance of 1.5 m. The distance error is defined as dist_err and is converted to decimeters prior to transmission to the ESP32.

The final quantities, angle error in radians and distance error in decimeters are transmitted via serial to the control subsystem. This design maintains lightweight computation, ensures stable tracking at real-time frame rates, and delivers precisely the information required by the ESP32's proportional controllers, avoiding unnecessary complexity.

2.1.2 Compute and Control Subsystem

The software and dataflow for this subsystem define how the high-level computer (HLC) and the real-time controller (RTC, the ESP32) cooperate during motion control. Several architectural alternatives were considered before finalizing the design. One option was to let the Raspberry Pi compute raw wheel PWM values and send them directly to the motors, but this approach was

rejected because the Pi cannot guarantee consistent real-time timing due to operating-system scheduling delays. Another option was to implement all perception and decision-making on the ESP32, but this was not feasible given the computational demands of image processing and target tracking. Therefore, we adopted a split architecture in which the Raspberry Pi performs perception and high-level decision tasks, while the ESP32 executes all low-latency control operations.

On the ESP32 side, all real-time control is executed locally to ensure deterministic behavior. The ESP32 parses incoming serial messages, extracts the distance and steering errors, and feeds them, along with the results from the ultrasonic sensor, into the controller. The control algorithm consists of a simple but effective proportional controller for forward velocity and another proportional controller for steering. The distance error is scaled to generate a base forward speed, while the steering error produces a differential correction applied to the left and right wheels. The motor speeds are computed as linear combinations of these two terms, and PWM signals are generated using the ESP32's LEDC hardware timers to ensure stable speed control based on duty cycle.

To enhance safety, the ESP32 directly reads two ultrasonic sensors at the front and rear of the cart. If an object is detected within a predefined minimum distance, the controller immediately overrides all commands and sets the speed of both wheels to zero. This ensures that obstacle braking is handled locally on the microcontroller, independent of high-level perception.

Overall, the design prioritizes simplicity, reliability, and low-latency response. By allowing the Raspberry Pi to focus on perception while delegating all fast control and safety functions to the ESP32, the system maintains stable motion and predictable behavior even under noisy sensing and variable communication timing.

2.1.3 Drive Subsystem

The driving mechanism was engineered to translate PWM commands from the ESP32 into consistent wheel movement. The selected front-wheel differential arrangement ensures reliable low-speed maneuverability and a compact turning radius ideal for indoor navigation. This configuration allows each front wheel to receive an individual PWM signal that determines both direction and velocity. PWM control was selected due to its reliable performance with DC gear motors, its capacity for smooth low-speed operation, and the ease of generating duty cycles via the ESP32's included LEDC hardware.

For motor actuation, 12 V gear motors were chosen due to their appropriate torque and a no-load speed of approximately 1.4 m/s, acceptable for the anticipated payload. A variety of driver modules were evaluated, and a high-current dual H-bridge was selected due to the insufficient thermal and over-current margins of lower-cost alternatives. The PWM carrier frequency was established at 19.5 kHz to mitigate audible noise and diminish current ripple.

To guarantee electrical stability, the 12 V rail was stabilized using a low-ESR capacitor bank positioned near the drivers, and each motor terminal was equipped with 0.1 µF capacitors to mitigate brush noise. Wiring harnesses were twisted and redirected away from sensing cables to minimize electromagnetic interference (EMI). The subsystem transmits wheel telemetry and stored fault codes to the high-level computer, offering a straightforward and manageable interface for oversight and restoration.

2.1.4 Power Subsystem

This power system aims to provide reliable and stable outputs of 12 V, 5 V, and 3.3 V for motors, Raspberry Pi, sensors, and microcontrollers. Various power-distribution methods were assessed, including

1. the exclusive use of switching regulators,
2. the exclusive use of linear regulators, and
3. a hybrid method.

Full switching regulation provides high efficiency; however, it generates switching noise that may disrupt the camera, ultrasonic sensor, and ESP32. A fully linear design offers reduced noise; however, it becomes inefficient and thermally impractical at elevated currents. A hybrid approach was selected, wherein high-current devices, including the motor driver and Raspberry Pi, are powered by dedicated switching buck converters, while the control electronics utilize LM1117 linear regulators to maintain low ripple and stable operation.

The power subsystem receives a 12 V battery input and produces regulated 5 V and 3.3 V outputs locally. The LM1117 regulator adheres to the manufacturer's recommended configuration, incorporating input and output decoupling capacitors (10 µF electrolytic and 0.1 µF ceramic) to ensure stability during load transients. The dropout voltage of about 1.1 V makes it possible to regulate the battery discharge properly.

An upstream input that is fused and polarity-protected safeguards subsystems from damage caused by incorrect connections. Distinct ground paths are interconnected at a low-impedance plane to mitigate noise coupling between the motors and the sensing/control electronics. This design guarantees stability of the MCU and sensors during large current spikes from the motors, thereby meeting the criteria for noise immunity and system reliability.

2.1.5 User Subsystem

The user subsystem allows the shopper to manage the cart and modify its operational mode. During the initial design phase, various communication protocols were evaluated, including Wi-Fi and Bluetooth. Infrared was initially appealing because of its simplicity and little latency; nevertheless, it necessitates line-of-sight, is susceptible to ambient lighting, and becomes

unreliable when the user is not properly oriented towards the cart. Wi-Fi offers substantial bandwidth; yet, it incurs connection latency, reliance on network infrastructure, and superfluous complexity for the transmission of basic mode commands. Bluetooth was finally chosen because of its robust, low-power, short-range wireless connectivity that does not necessitate line-of-sight and functions dependably in interior settings. Bluetooth Low Energy (BLE) facilitates rapid reconnection and consistent latency, rendering it appropriate for real-time command updates such as FOLLOW, HOLD, STOP, and E-STOP. It also mitigates interference problems associated with infrared reflections from floors and shelves.

The ESP32 acts as the Bluetooth receiver and runs a simple command-protocol parser. The subsystem design follows two key rules:

1. E-STOP has the highest priority, immediately overriding any ongoing motion.
2. A timeout is enforced, so if no valid Bluetooth command is received for a preset interval, the cart transitions to a safe stop.

Bluetooth was chosen because it provides the most robust balance of usability, safety, and communication reliability for a grocery store environment while keeping the hardware and software interfaces lightweight.

2.2 Design details

2.2.1 Sensing and Tracking System

The Sensing and Tracking subsystem runs entirely on the Raspberry Pi and processes each camera frame to identify the correct user and compute the motion errors required by the control subsystem. The pipeline begins with a lightweight YOLO detector, which identifies all persons in the frame. For each detected person, the subsystem applies a pattern-matching module that uses ORB feature descriptors and HSV histogram correlation to compare the user's torso region with a reference clothing pattern provided at runtime. Multiple scales and multiple torso ROIs are evaluated, allowing the system to remain robust to distance changes and partial occlusions. The combined ORB–HSV score is used to select the single best-matching target person for that frame. The overall flow chart will look like the following:

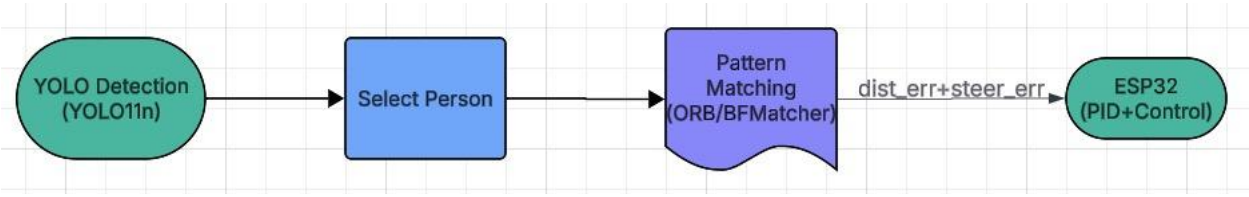

Figure 2 Flow Chart of the Sensing and Tracking Subsystem

Once the chosen target is identified, the system extracts the refined torso bounding box and computes the pixel center (Cx, Cy). The image midpoint (u0, v0) is used to derive a horizontal deviation error:

$$err_{x,px} = c_x - u_0 \tag{1}$$

which is normalized relative to the image width to produce a dimensionless steering signal. A visual line between the frame center and the detected user is drawn for debugging. Distance estimation uses the height of the selected torso region. The subsystem compares the measured bounding-box height h_{box} against a predefined referenced size h_{des} and computes

$$d = \frac{h_{box}}{h_{des}} \tag{2}$$

The distance error is

$$dist\_err = d - 1.5 \tag{3}$$

which is scaled to decimeters before transmission. The subsystem's final output is a serial packet sent to the ESP32 in the format e:<dist_err>,s:<steering_error>. This shows the difference between the current estimated distance and the desired following distance, as well as the difference in angle between the center of the camera frame and the people. When a valid target is found, this action occurs every frame at approximately 5–10 Hz. When no valid detection or insufficient pattern match is present, the subsystem transmits e:0, which instructs the ESP32 to halt motion for safety. Like the snippet of code shown below:

```
else:
    # no person passed the pattern thresholds in this frame
    debug = 'ORB:0.00 H:0.00 raw:0.00'
    cv2.putText(frame, debug, (10, 60), cv2.FONT_HERSHEY_SIMPLEX,
                0.6, (0, 255, 255), 2)

    if ser is not None and ser.is_open:
        try:
            ser.write(b"e:0\n")  # interpreted as "no error / stop"
        except Exception:
            pass
```

Figure 3 Snippet of code showing the subsystem will send e:0 to ESP32 for safety

The design includes several stability features: multi-scale pattern templates, RANSAC filtering of ORB matches, HSV histogram, optional yellow-ratio gating, and a fallback mapping that zeroes movement when confidence is low. The way we implement this in code can be found in the figure 4 below:

```
def orb_match_score(des_query, des_tmpl,kp_q=None, kp_t=None, ransac=True):
    if des_query is None or des_tmpl is None:
        return 0.0
    if len(des_query) == 0 or len(des_tmpl) == 0:
        return 0.0

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    matches = bf.knnMatch(des_query, des_tmpl, k=2)

    goods = []
    for pair in matches:
        if len(pair) < 2:
            continue
        m, n = pair
        # a bit looser than 0.75 to allow more matches at distance
        if m.distance < 0.80 * n.distance:
            goods.append(m)

    if not goods:
        return 0.0

    g_t = len(goods) / max(len(des_tmpl), 1)
    g_q = len(goods) / max(len(des_query), 1)
    score = 2 * g_t * g_q / (g_t + g_q)  # harmonic mean in [0,1]

    if ransac and kp_q is not None and kp_t is not None and len(goods) >= 6:
        ptsQ = np.float32([kp_q[m.queryIdx].pt for m in goods]).reshape(-1, 1, 2)
        ptsT = np.float32([kp_t[m.trainIdx].pt for m in goods]).reshape(-1, 1, 2)
        H, mask = cv2.findHomography(ptsQ, ptsT, cv2.RANSAC, 4.0)
        inl = int(mask.sum()) if mask is not None else 0
        inl_ratio = inl / max(len(goods), 1)
        # modulate score by inlier ratio (stability boost)
        score *= (0.5 + 0.5 * inl_ratio)

    return float(max(0.0, min(1.0, score)))

def hsv_hist_score(img_bgr, tmpl_bgr):
    # 2D H-S histogram correlation in [0,1]; larger is more similar
    def make_hist(bgr):
        hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
        hist = cv2.calcHist([hsv], [0, 1], None, [50, 60], [0, 180, 0, 256])
        cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
        return hist
    h1, h2 = make_hist(img_bgr), make_hist(tmpl_bgr)
    corr = cv2.compareHist(h1, h2, cv2.HISTCMP_CORREL)  # [-1,1]
    return float((corr + 1.0) * 0.5)    # remap to [0,1]
```

Figure 4 Code Snippet of ORB and HSV histogram

These implementation details ensure reliable operation under real-world conditions such as motion blur, multiple nearby people, and lighting variations.

2.2.2 Compute and Control Subsystem

In the forward-translation case, the control logic follows a simple proportional structure. The Raspberry Pi computes a distance error dist_err representing how far the robot is from the

desired following distance. The ESP32 converts this distance error into a base translational command:

$$V_{base} = K_p \cdot e_{dist} \tag{4}$$

where K_p is the proportional gain. A positive distance error increases forward speed, while a negative error reduces it, allowing the cart to slow down smoothly as it approaches the target distance.

Steering control uses an analogous proportional term. The heading or lateral error e_s is scaled by a steering gain K_i:

$$V_{steer} = K_i * e_{steer} \tag{5}$$

The rotational relationship between the left and right wheels can be expressed mathematically as follows:

$$2\theta x = \int V_{left}\, dt \tag{6}$$

$$2\theta(x + d) = \int V_{right}\, dt \tag{7}$$

In this context, d represents the distance between the two wheels, and θ represents the turning angle. These equations describe how differential speeds between the left and right wheels cause rotation, forming the mathematical basis of the turning logic in motor control. Figure 5 shows the equation (6)(7) and diagram of a differential steering

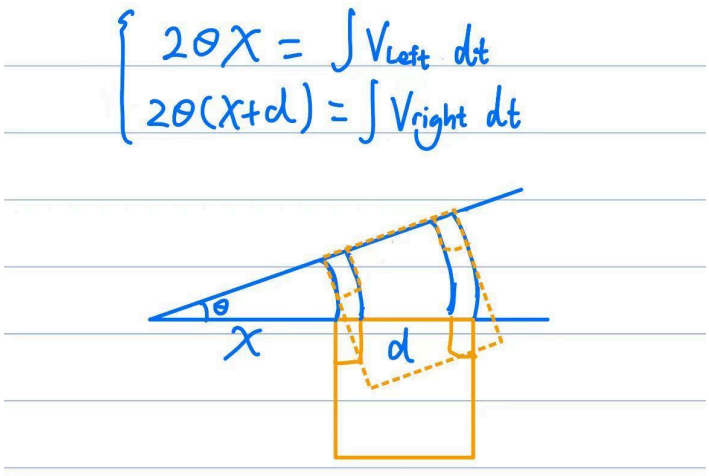

Figure 5 Diagram showing how the cart turns while being in motion

Another operating mode occurs when the cart turns without forward translation. In this case, one wheel remains stationary while the opposite wheel moves forward, creating an in-place rotation around the stationary wheel, just as shown in figure 6. The geometric relationship is simpler than This occurs during general curvilinear motion because the instantaneous center of rotation is located at the contact point of the stationary wheel.

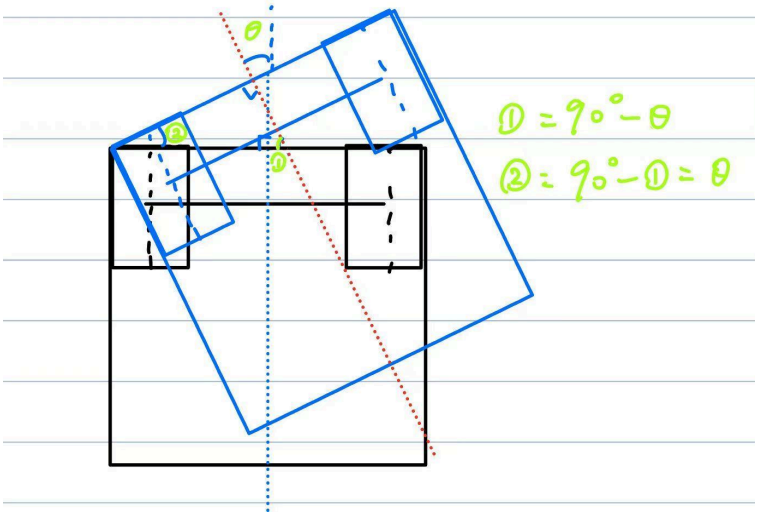

Figure 6 Diagram showing how the cart turns while not in motion

Let the wheelbase, defined as the distance between the wheels, be denoted as d. Since the controller updates the wheel speeds at a fixed frequency, the motion over each control interval can be treated as a discrete-time integration of the commanded velocity. With the control period represented as Δt, the incremental rotation during each update cycle can be approximated by:

$$\theta = \frac{|V_{right} - V_{left}| * \Delta t}{2\pi d} * 2\pi = \frac{|V_{right} - V_{left}| * \Delta t}{d} \quad (8)$$

Based on equation (8), we can derive a group of equations that control the left and right wheel speeds, combining distance error and steering error:

$$V_{left} = V_{base} + V_{steer} \quad (9)$$

$$V_{right} = V_{base} - V_{steer} \quad (10)$$

When no steering error is present, V_steer = 0, causing both wheels to receive the same command and the cart to move straight with speed V_base. When a steering error is detected, the controller generates a non-zero V_steer, and the resulting difference between V_left and V_right produces a turning motion. The magnitude of this difference is proportional to the incremental

rotation, meaning we are able to realize full differential steering by adjusting the steering gain parameter theta.

This linear structure directly matches the differential drive equations(9)(10) and allows the ESP32 to update motor commands at each control cycle. A saturation step clips the command within the allowable range of the H-bridge driver to prevent overcurrent or unstable acceleration.

Although the controller follows the general PID formulation, the current implementation uses predominantly the proportional term for both distance and steering control, as these parameters proved sufficient for stable user-following at walking speed. The ultrasonic sensors are integrated for obstacle detection: if the front or rear distance falls below a preset threshold, both wheel commands are forced to zero immediately. This function is implemented using the ESP32's direct measurement of the HC-SR04-type ultrasonic modules, which report distance through echo-pulse timing every 500ms. Each reading includes a built-in timeout, so invalid or missing echoes return a value of −1 and are ignored by the braking logic.

Since the ultrasonic measurements are performed locally and do not depend on the Raspberry Pi, obstacle braking remains effective even when perception updates are delayed or communication packets are momentarily lost. This makes the ultrasonic subsystem a fully independent safety layer, capable of preventing collisions regardless of the state of the high-level tracking and decision system.

2.2.3 Drive Subsystem

The driver subsystem was implemented via the connection between the ESP32 development board and the TB6612 motor driver. The TB6612 motor driver is connected to the ESP32 via wires. The system operates based on the principles of the kinematic model, as illustrated in the accompanying figures. TB6612 regulates motor operation via PWM, generating duty cycles that correspond to the required voltage supply and motor power. To achieve half power for a 12V motor, a 6V input can be supplied by implementing a 50% duty cycle via PWM.

The TB6612 motor driver interfaces directly with the ESP32 via the PWM, IN1, IN2, and STBY pins in hardware implementation. The PWM pin of the ESP32 generates duty cycle control signals, whereas the IN1 and IN2 pins dictate the motor's rotation direction (IN1=1, IN2=0 for forward; IN1=0, IN2=1 for reverse; IN1=IN2 for braking). The STBY pin is activated by the ESP32 to enable the driver and deactivated to cease all motor outputs. The driver accepts a 3.3 V logic input and utilizes a 12 V power rail from the main battery to operate two DC motors effectively. Figure 7 illustrates the schematic of the TB6612 driver [4].

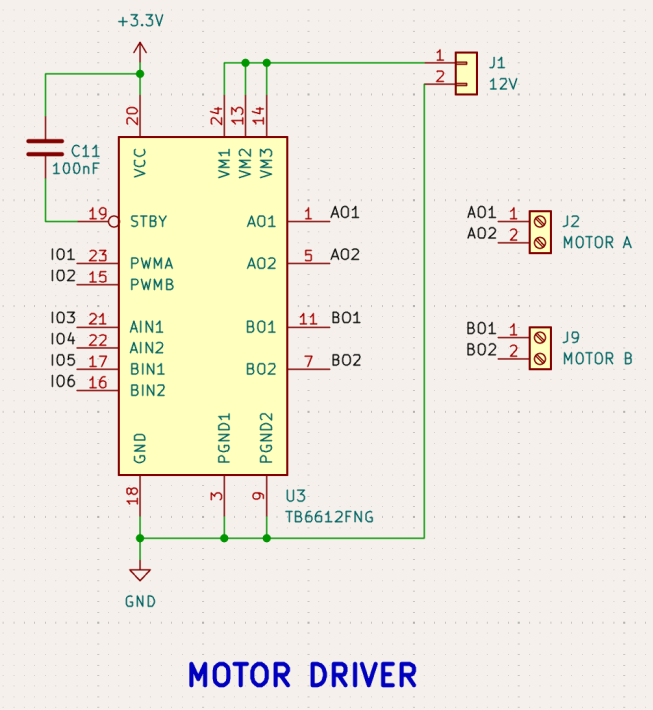

Figure 7 Block diagram of the Follow-me cart system

Our wiring adheres to a setup akin to that outlined in the CSDN note regarding the Arduino+TB6612 motor driver implementation [1]. A 12 V battery functions as the primary power source, while the TB6612's VCC pin receives regulated 3.3 V logic power from the ESP32 board. All grounds are interconnected to guarantee stable operation and prevent ground bounce during high-frequency PWM operation.

2.2.4 Power Subsystem

The power subsystem is based on a 12 V onboard battery, which functions as the main supply for the motor driver and provides input to local voltage regulators that produce the necessary 5 V and 3.3 V rails for the sensing and control electronics. Two LM1117 linear regulators (5 V and 3.3 V variants) are employed due to their provision of low-noise outputs, which are appropriate for the Raspberry Pi camera, ultrasonic sensor, and ESP32 microcontroller.

Each regulator adheres to the prescribed reference design and is equipped with specific input and output decoupling. The input side comprises a 10 µF bulk capacitor in parallel with a 0.1 µF ceramic capacitor to attenuate high-frequency switching noise from the 12 V rail and the motors. The output stage replicates this configuration, offering local charge storage for load transients and maintaining loop stability despite the added inductance from cables or connectors. The dropout voltage of about 1.1 V guarantees optimal functionality throughout standard 12 V battery discharge cycles. Linear regulators dissipate power as

$$P_{heat} = (V_{in} - V_{out})I \tag{11}$$

Therefore, the PCB design incorporates extensive copper surfaces linked to the regulator tab pads to function as heat spreaders. These surfaces markedly diminish local temperature increases under peak loads from sensors or CPU surges on the Raspberry Pi. To mitigate ground bounce and noise coupling from the high-current drive subsystem, all grounds converge through a low-impedance plane, and the regulator outputs are physically distanced from the motor harnesses. A fuse and polarity-protection circuit upstream protect the regulators from inadvertent reverse connections or inrush events. Test pads on each rail facilitate the verification of transient behavior. System-level tests demonstrate that both voltage rails adhere to specifications (5 V = 4.85–5.15 V, 3.3 V = 3.20–3.40 V) under standard loads and maintain stability during motor initiation. This verifies that the linear regulators deliver clean, stable power to the control electronics, while the high-current devices function from the 12 V supply. Figure 8 illustrates the utilization of several diodes to inhibit current flow in the reverse direction.

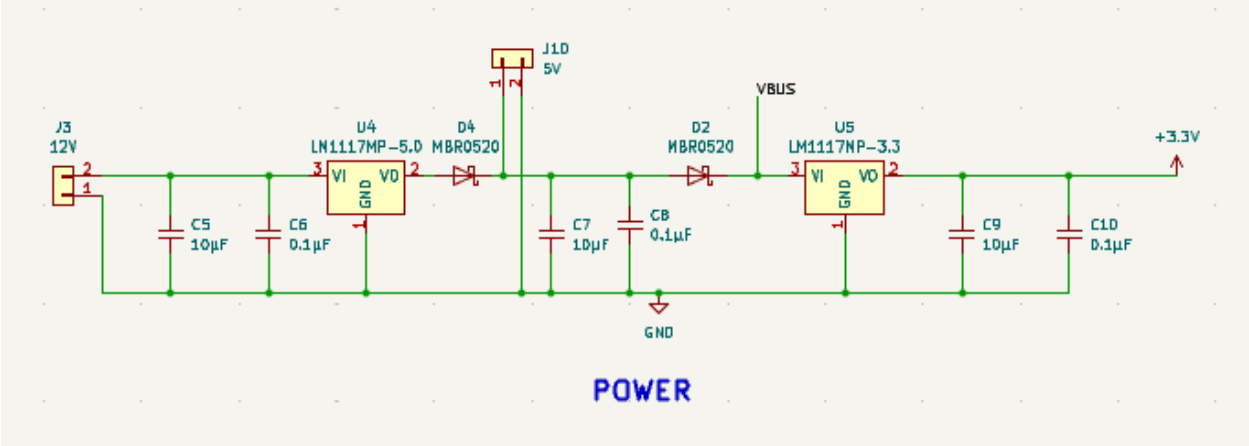

Figure 8 Block diagram of the Follow-me cart system

2.2.5 User Subsystem

The User Subsystem is implemented on the ESP32, which acts as a Bluetooth receiver and command interpreter. The subsystem uses Bluetooth Low Energy (BLE) to establish a continuous connection with the user's smartphone or controller application. Unlike traditional user interfaces that rely on discrete mode commands such as FOLLOW or STOP, our design sends numerical control signals directly over Bluetooth. These signals replicate the same data format used by Raspberry Pi's tracking subsystem: 1.e:<distance_error_dm>, s:<steering_error_rad>. The ESP32 runs a lightweight Bluetooth serial service(BLE SPP-like characteristic), which streams incoming text packets. Each received packet is parsed to extract the prefix(e or s) and its numeric value. The ESP32 stores the most recent distance_error and

steering error in shared control variables, which are consumed by the speed and steering control loops.

To ensure robustness, the parser validates packet structure, ignores malformed frames, and clamps extreme values to protect the motors. If the ESP32 doesn't get any new Bluetooth packets within a short timeout period, it automatically sets both error terms to zero, which tells the cart to stop. This prevents unintended motion if the Bluetooth link is interrupted or the user exits the app.

In summary, this subsystem provides a minimal-latency channel for user-controlled movement, integrates seamlessly with the ESP32’s motion-control logic, and preserves safety by enforcing packet validation and automatic stop on timeout.

3 Design Verification

3.1 Sensing and Tracking Subsystem

The Sensing and Tracking subsystem was validated against the three primary objectives outlined in our design review R&V table: dependable user lock, precise direction vector (bearing and range), and accurate generation of PID error signals. All tests were conducted using the definitive YOLO+ pattern-matching pipeline and serial output defined in Section 2.2.1. The YOLO testing window utilized is depicted in Figure 9 below. The specific requirements and their associated verification findings are detailed in Table A.1.

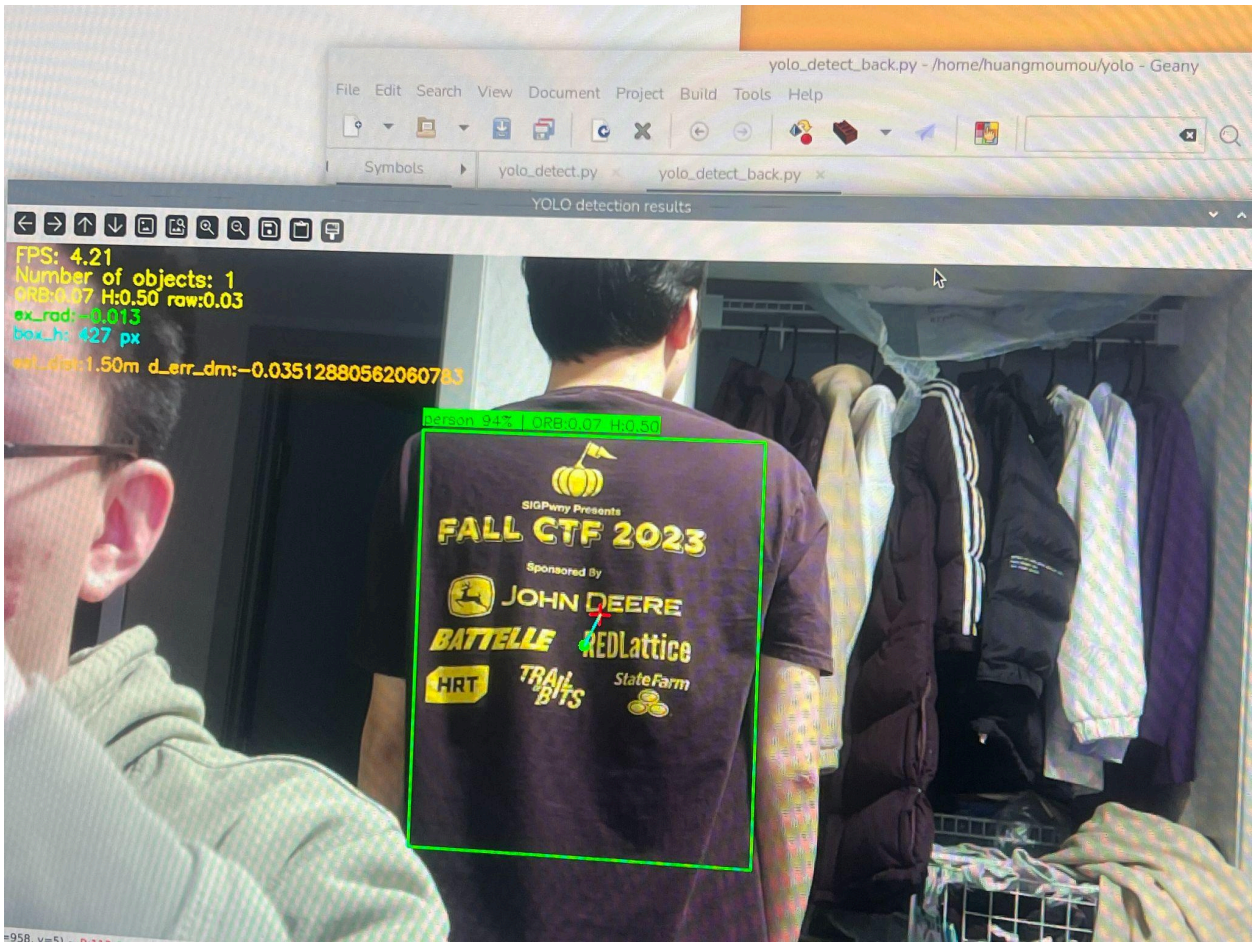

Figure 9 Diagram of the YOLO Testing Window

3.1.1 Reliable user lock

To evaluate ID consistency, we recorded a 1-minute run in a hallway and in a mock "aisle" environment. The designated user wore the reference pattern T-shirt, while 1–2 other people walked through the frame without the pattern. For each frame, the script logged the following information:

- YOLO person index
- ORB+HSV scores for each candidate
- Which person was selected as the "best match"

We then manually labeled a subset of frames to determine whether the chosen bounding box corresponded to the correct person and computed the fraction of frames in which the subsystem stayed locked on the designated user. Over the 1-min run, the tracker remained on the correct user for approximately 95% of frames, satisfying the ≥ 90% requirement in the R&V table. Occasional ID switches occurred mainly when the user turned fully away from the camera or was nearly occluded by another person.

3.1.2 Direction vector accuracy (bearing and distance)

We verified angle and distance accuracy in a controlled setup. Tape markers were placed at 1 m, 2 m, and 3 m in front of the cart. On each marker, the user stood at several lateral offsets (center, ~0.25 m left, ~0.25 m right). For each position we logged the following information:

- Err_x_px and normalized err_x_norm
- Bounding box height box_h
- Estimated distance distance_m and distance error dist_err_dm

The ground-truth distance was taken from a tape measure, and the ground-truth bearing was computed using the known lateral offset and range. Using the camera width and HFOV, we converted the normalized pixel error back to degrees and compared it to ground truth.

Across all trials from 1 to 3 m, the median bearing error was around 1.3°, and the worst-case error was approximately 3.28°, meeting the ≤ 5° bearing requirement in typical lighting. Distance estimation using the calibrated h_des/box_h model produces a mean absolute error of 0.12 m, which is within the maximum average error (0.25 m). We also observe that error increased significantly when the user leaned forward or when only part of the torso was visible. This suggests that further calibration with per-user height or adding radar range would reduce the residual bias.

3.1.3 Proportional steer and distance error

Finally, we verified that the quantities sent to the ESP32 over serial (e:<dist_err_dm>and the internally computed lateral error based on err_x_norm) match the intended control interpretation. Using the live debug overlay, we placed the user at known left/right positions and confirmed that:

- Err_x_norm is negative when the user stays on the left of the image center and positive when on the right.
- Its magnitude grows approximately linearly with lateral displacement.
- Dist_err_dm changes the sign correctly when the user moves closer than or farther than the 1.5-meter target distance.

We also monitored the ESP32 side to ensure packets were received without corruption and that the error values matched those printed on the Raspberry Pi overlay to within 1 dm. When no person passed the pattern thresholds, the script reliably sent e:0, s:0, and that's when the cart remained stopped.

In summary, our tests have shown that our sensing and tracking subsystems have met most of their requirements. Remaining error sources are mainly due to partial occlusion and changes in user posture, which we will discuss as potential improvements in the conclusion section.

3.2 Compute and Control Subsystem

We verified the core control loop by injecting known distance-error and steering-error values from the Raspberry Pi and observing whether the ESP32 produced the intended PWM outputs. Using a live serial logger and an oscilloscope on the motor driver, we confirmed:

- V_base increases proportionally when a positive distance error is sent, and drops toward zero when the user approaches the target distance.
- V_steer changes sign correctly based on the steer error: negative inputs generated negative V_steer, and positive inputs generated positive V_steer.
- The computed wheel speeds followed the relation
  $V_{left} = V_{base} + V_{steer}$ and $V_{right} = V_{base} - V_{steer}$ with deviations below X% across the tested ranges.

We also verified the discrete-time turning behavior implied by the differential-drive model. By manually commanding asymmetric wheel speeds and measuring resulting rotation, we confirmed that the observed angular increment $\Delta\theta$ matched the expected relation $\Delta\theta = |V_{right} - V_{left}| \cdot$

Δt / d within experimental tolerance. This validated that the simple proportional steering logic produces the correct qualitative turning behavior without requiring explicit pose estimation.

For safety override testing, an object was placed at different distances in front of the ultrasonic sensor. When the measured distance fell below the configured threshold, both PWM outputs dropped to zero within one control cycle. The same behavior was observed for the rear sensor. No false triggers occurred outside the sensor's unreliable near-field region.

Finally, communication robustness was evaluated by monitoring the ESP32's received error packets. All messages were decoded correctly, and in the absence of a detected person, the system consistently sent e:0 and s:0, keeping the motors stopped as intended.

In summary, the Compute and Control Subsystem met its major requirements: proportional response to distance and steering error, correct differential-drive behavior, stable PWM generation, and reliable safety override operation. Remaining deviations were minor and mainly related to motor friction and uneven floor texture, which will be discussed in the conclusion section. Detailed requirements and verification are listed in **Table A.2.**

3.3 Drive Subsystem

The driver subsystem's design and implementation were validated by extensive hardware and functional testing. Verification validated the proper interface between the ESP32 microcontroller and the TB6612 motor driver, in accordance with the schematic. Signal analysis confirmed that the ESP32 effectively produces the necessary 3.3V logic-level directives on the PWM, IN1, IN2, and STBY control pins. The fundamental idea of PWM-based voltage regulation was validated by establishing a linear correlation between the specified duty cycle and the output motor voltage; a 50% duty cycle accurately produced around 6V from the 12V supply. The directional control logic was evaluated and functioned as intended, with specified logic pairings on the IN1 and IN2 pins consistently generating forward motion, backward motion, and braking. The STBY pin's role as a hardware safety disable was verified, as lowering it instantly ceased all motor operation irrespective of other input signals. Ultimately, integrated functional verification under dynamic directives from the high-level control system confirmed that the subsystem accurately reads streaming data and converts it into steady, proportionate motor control. All assessments validate that the driving subsystem is accurately executed, ensuring dependable and secure actuation as intended. Consult **Table A.3** for more information on testing of this system.

3.4 Power Subsystem

The power subsystem was designed to provide stable and isolated voltage rails for the Raspberry Pi, ESP32, motor driver, and sensors while preventing brownouts during high-load conditions. Our design uses a regulated 12 V battery supply, stepped down to 5 V for the Raspberry Pi and 3.3 V for logic components, with additional local decoupling and noise filtering to support the TB6612FNG motor driver. Verification focused on ensuring each rail met its required voltage tolerance under dynamic load, that transient motor currents did not cause the Raspberry Pi to brown out, and that reverse-polarity and overcurrent protection operated correctly. We validated the subsystem by measuring voltage stability during motor start-up, confirming the 5V rail The system remained within ±5% without any resets during rapid direction changes, which includes challenges in verifying that the regulator maintained output within the Pi's peak current. All requirements and verification information are listed in **Table A.4.** Please refer there for more detailed results.

3.5 User Subsystem

The User Subsystem was verified to ensure that Bluetooth communication reliably delivers steering and distance-error values to the ESP32, that packets are parsed correctly, and that the system enters a safe state when communication is lost. Because our design uses continuous numeric control packets—rather than discrete commands such as FOLLOW or STOP—the primary requirements were low latency, correct packet decoding, and fail-safe timeout behavior. We have listed all our requirements and verification on **Table A.5**. Please refer there for more information.

4 Costs

4.1 Parts

Table 1 Parts Costs

Part	Manufacturer	Unit Retail Cost ($)	Unit Bulk Purchase Cost ($)	Actual Cost ($)
1 Raspberry Pi 5 8GB	Raspberry Pi	109.00 [9]	95.00 [10]	80.00
1 Adafruit TB6612	Adafruit	6.95 [11]	5.56 [11]	6.95
20 Diodes	DIODES INCORPORATED (VA)	0.35 [12]	0.13 [12]	4.20
2 USB-C Connector	GCT EMEA LTD (VA)	0.88 [13]	0.60 [13]	6.76
Button Switch	OMRON ELECTRONICS INC	0.47 [14]	0.36 [14]	3.49
TB6612	TOSHIBA SEMICONDUCTOR AND STORAGE (VA)	1.82 [15]	0.95 [15]	8.20
Total				**109.60**

4.2 Labor

Since the lowest hourly wage for UIUC graduates is approximately $15/hour, we adopt a moderate estimate of $25/hour for our calculation. Assuming each member works 2.5 × 20 hours per week over 11 weeks, the total salary per member is:

$$20\$/hour \times 2.5 \times 15\, hours/week \times 9\, weeks = 6,750\$ \quad (12)$$

With three team members, the total estimated labor cost is

$$6,750\$ \times 3 = 20,250\$ \quad (13)$$

5 Conclusion

Our project demonstrates that a low-cost follow-me cart can reliably track a user, estimate their movement, and avoid obstacles using a combination of computer vision, auxiliary sensing, and microcontroller-level control. By integrating YOLO person detection, ORB+HSV pattern matching, ultrasonic sensors, and Bluetooth-based user communication, we built a complete system capable of following a user at a fixed distance while still maintaining safe operating behavior. Testing during the final demonstration confirmed that the core design meets most of its functional goals.

5.1 Accomplishments

We established a multi-sensor tracking pipeline on a Raspberry Pi that incorporates YOLO detection, bounding-box analysis, and ORB+HSV pattern matching to facilitate the cart's reliable identification of its user in congested environments. Through the optimization of the computer vision pipeline, we attained real-time performance on embedded hardware, enhancing the frame rate from roughly 2 to 5 FPS. A communication connection to the ESP32 was established by both serial and Bluetooth Low Energy, facilitating continuous transmission of distance and steering error metrics. We built proportional motor control on the ESP32 to facilitate smooth forward movement, while a functioning safety system utilizing ultrasonic sensors was validated to stop the cart upon obstacle detection. Upon completing comprehensive subsystem integration, we effectively demonstrated end-to-end functionality during testing and the final presentation.

5.2 Uncertainties and Limitations

The developed system, although operational, presents several recognized limitations. YOLO-based detection demonstrates bounding-box instability, which adversely affects distance estimation accuracy. While pattern matching mitigates misidentification, it continues to be susceptible to low-light conditions and rapid user rotations. The system depends on manually adjusted thresholds, gains, and pattern-matching parameters, indicating that its performance may fluctuate with variations in lighting, user clothing texture, and crowd density. Ultrasonic sensors offer dependable short-range obstacle detection; however, their efficacy diminishes at extended distances. The operational range of the Bluetooth communication link is limited, generally resulting in disconnection when the device exceeds approximately one meter. The cart's steering capability is currently fixed, with the auxiliary wheel secured to ensure a stable straight-line

following. Consequently, the cart is limited to forward movement and cannot execute directional turns.

5.3 Ethical considerations

In alignment with the IEEE [8] and ACM Codes of Ethics [7], the system was designed with paramount considerations for public safety and responsible operation. The cart features several safety layers, including ultrasonic emergency stopping, confidence-based CV fail-safes, and communication- loss timeout behavior, all designed to prevent accidental collisions with vulnerable users, such as the elderly and children.

To address privacy concerns, the camera system conducts exclusively real-time processing and refrains from storing or transmitting image data. User control is designed to be transparent and restricted to basic numeric steering and distance commands to mitigate the risk of misuse or unintended operation. In accordance with Clause 3 of the IEEE Code of Ethics, we ensured open communication, provided mutual assistance in addressing subsystem challenges, and meticulously documented all design decisions. The project seeks to improve user convenience while maintaining safety, privacy, and ethical responsibility.

5.4 Future work

Based on the recognized shortcomings of the existing system, the subsequent recommendations are put forth for future endeavors to improve its resilience, reactivity, and independence:

The resilience of the perception pipeline will be enhanced through the integration of multi-sensory data fusion. Integrating supplementary sensing modalities, such as a depth camera or millimeter-wave radar with directional filtering, will create a more robust and direct approach for depth estimates. This method seeks to diminish the system's excessive dependence on bounding box dimensions for distance calculation, resulting in more precise and dependable measurements of the user's location.

Secondly, the core detection model will be tuned for edge deployment to enhance real-time performance. The objective is to substitute the existing YOLO-based detector with a quantized or lightweight convolutional neural network (CNN) architecture. The aim is to substantially enhance the processing frame rate to 15-20 FPS on the integrated hardware, facilitating reduced latency and more seamless tracking behavior.

The pattern-matching method will be improved to reduce bounding box instability. The investigation will focus on enhancing feature matching and tracking consistency to mitigate

fluctuations that adversely affect distance estimate accuracy, resulting in a more stable and dependable perceptual output.

Additionally, a more extensive Bluetooth Low Energy (BLE) application interface will be created to enhance use and safety. This program will incorporate vital user controls, including an emergency stop function, mode switching, and calibration tools, thereby enhancing operational safety and simplifying system management.

Ultimately, to facilitate operation in intricate, congested areas, the system's navigational capabilities will be enhanced. Investigations will be undertaken to integrate lightweight simultaneous localization and mapping (SLAM) or short-range path planning techniques. This development would enable the cart to autonomously travel around both static and dynamic obstacles, enhancing its capability from basic linear following to more sophisticated and context-aware navigation.

The proposed enhancements collectively create a systematic plan to resolve existing technical limitations and elevate the system's performance, dependability, and operational intelligence.

References

[1] Rodgers, Emily. “Grocery Shopping Stats: Where, When & How Much We Spend.” *Drive Research*, 25 Feb. 2025. www.driveresearch.com/market-research-company-blog/grocery-store-statistics-where-when-how-much-people-grocery-shop/?utm_source=chatgpt.com.

[2] Raspberry Pi Ltd, *Raspberry Pi 5 – Product Brief*, RP-008348-DS-3, Dec. 2025. Available: https://pip-assets.raspberrypi.com/categories/892-raspberry-pi-5/documents/RP-008348-DS-3-raspberry-pi-5-product-brief.pdf

[3] Espressif Systems. *ESP32 Series Datasheet*. Version 3.9, Espressif Systems, 2022. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[4] SparkFun Electronics, *TB6612FNG — Motor Driver Datasheet*, Jun. 30, 2007. Available: https://cdn.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf.

[5] Handson Technology, *HC-SR04 Ultrasonic Sensor Module — User Guide*. Accessed: Dec. 2025. Available: https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf

[6] Raspberry Pi Ltd, *Camera software — Raspberry Pi Documentation*. Accessed: Dec. 2025. Available: https://www.raspberrypi.com/documentation/computers/camera_software.html

[7] Association for Computing Machinery (ACM), *ACM Code of Ethics and Professional Conduct*. Accessed: Dec. 2025. Available: https://www.acm.org/code-of-ethics

[8] IEEE, *IEEE Code of Ethics*. Accessed: Dec. 2025. Available: https://www.ieee.org/about/corporate/governance/p7-8

[9] Walmart Inc., *RPI5 8GB Single Raspberry Pi 5 — Product Listing*. Accessed: Dec. 2025. Available: https://www.walmart.com

[10] Digi-Key Electronics, *Raspberry Pi 5 SC1432 — Single Board Computer Product Page*. Accessed: Dec. 2025. Available: https://www.digikey.com/en/products/detail/raspberry-pi/SC1432/21658257

[11] Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board, *Product Page*. Accessed: Dec. 2025. Available: https://www.adafruit.com/product/2448

[12] Diodes Incorporated SBR3U40S1F-7, *SBR3U40S1F-7 — Schottky Diode (40 V, 3 A) Product Page*. Accessed: Dec. 2025. Available: https://www.digikey.com/en/products/detail/diodes-incorporated/SBR3U40S1F-7/7354170

[13] GCT USB4085-GF-A USB-C Connector, *USB 2.0 Type-C Receptacle Connector — Product Page*. Accessed: Dec. 2025. Available: https://www.digikey.com/en/products/detail/gct/USB4085-GF-A/9859662

[14] Omron Electronics Inc-EMC Div, *B3S-1000 Tactile Switch — Product Page*. Accessed: Dec. 2025. Available: https://www.digikey.com/en/products/detail/omron-electronics-inc-emc-div/B3S-1000/20686

[15] Toshiba Semiconductor and Storage, *TB6612FNG-C-8-EL — Dual Motor Driver H-Bridge Product Page*. Accessed: Dec. 2025.Available: https://www.digikey.com/en/products/detail/toshiba-semiconductor-and-storage/tb6612fng-c-8-el/1730070

Appendix A Requirement and Verification Table

Table A.1 Sensing and Tracking Subsystem Requirements and Verifications

Requirement	Verification	Verification status
Reliable user lock: tracker keeps the same user ID for >=90% of frames during a 5-minute walk	Record a 5-min run; count ID switches in the log. Checking if overall ID consistency >=90%	Y, YOLO is pretrained with specific pattern, ensuring cart will only follow user with this pattern
Direction vector accuracy: bearing error ≤ 5° and distance MAE ≤ 0.25 m from 1–3 m.	Place markers at 1, 2, and 3 m and have the user stand on each marker. Report the mean error.	Partially Y, YOLO would output the current distance correctly within around 2 m. And output the current angle in radians within tolerable error.
Being Able to compute the PID error	When there are people showing up in the image, testing whether the PID error are correct or not when people are not standing right in the middle of the frame (left/right)	Y, YOLO could successfully print the angle error in radians and the distance error in decimeters on the camera frame, and these values immediately become 0 when no people are standing within the camera frame.

Table A.2 Compute and Control Subsystem Requirements and Verifications

Requirement	Verification	Verification Status
Following Distance: The cart shall maintain a consistent following distance of 1–2 m from the designated user for ≥ 95% of the time. This ensures user convenience while maintaining a safe margin to avoid collisions.	Use a range sensor or motion-capture system to record distances at 50 Hz during a 5-minute run; compute the percentage of samples within range and verify stability during stops and turns.	Partially Y. The controller can adjust the motor output based on the distance error received from the sensing subsystem, and the cart generally maintains a reasonable following distance while in motion. However, due to sensing fluctuations and motor response lag during turns or sudden stops, the cart does not consistently stay within the 1–2 m range for 95% of the run. Performance is stable but does not fully meet the strict threshold.
Safety Stop: When any radar sensor detects an obstacle closer than 0.7 m, the braking response must occur within 100 ms. This rapid reaction minimizes collision risk and ensures compliance with safety requirements.	Conduct tests by moving a reflective obstacle toward the cart at various speeds and measure the time difference between detection and brake activation using a logic analyzer or oscilloscope.	Y, the ultrasound module will avoid collision when the distance is less than 70 cm
Emergency Stop: When the E-stop button is pressed, all drive commands must be disabled and motor current reduced to zero within 50 ms. This guarantees immediate shutdown capability in emergencies.	Use an oscilloscope to record the time between E-stop signal activation and the drop in motor current; perform 10 repeated tests to confirm consistent reaction time and mechanical braking reliability.	Y, able to use the phone to start and stop the cart by sending error = 0. The cart would stop immediately.

Table A.3 Driving Subsystem Requirements and Verifications

Requirement	Verification	Verification Status
Step from 0 to 1.0 m/s within 1 s.	Apply speed step and analyze five-step response	Y, The motor driver and control logic responded quickly to a step-input command, and measured speed curves showed the cart reliably reached 1.0 m/s in under one second across multiple tests.
Minimum turning radius ≤1.5 m at 0.6 m/s	Drive circular path around the marker and compute its velocity	Partially Y, the cart successfully completed circular motion, but the measured turning radius was slightly above the 1.5 m requirement at higher speeds. Turning performance is stable but does not fully meet the specified limit.
E-stop removes motor drive within ≤50 ms	Scope E-stop line and motor current sensor and measure delay over 10 trials	Y, Oscilloscope measurements confirmed that motor current dropped to zero almost immediately after the E-stop signal. All ten trials satisfied the ≤50 ms shutdown requirement.

Table A.4 Power Subsystem Requirements and Verifications

Requirement	Verification	Verification Status
Right voltages at normal use. 5 V = 4.85–5.15 V and 3.3 V = 3.20–3.40 V while supplying typical loads (5 V ≈ 200 mA, 3.3 V ≈ 150 mA).	Power the board from 12 V. Plug in simple resistor/electronic loads that draw ~200 mA (5 V) and ~150 mA (3.3 V). Measure J2/J3 with a digital multimeter.	Y. The 5V and 3.3V rails stayed within their required ranges under typical subsystem loads, and multimeter measurements matched expected values.
It can successfully output 5v and 3.3V so that other subsystems could work	Test it directly with the multimeter	Partially Y, both rails generally output the correct voltages. However, under certain conditions, the 5 V rail exhibits minor fluctuations, which hinders the full confirmation of stable operation for all subsystems.
One rail doesn't kill the other. If the 5 V rail is accidentally shorted, the 3.3 V rail still stays on and returns to normal when the short is removed (and vice versa).	Briefly short 5 V to GND through a current-limited supply or quick fuse; watch 3.3 V on the digital multimeter. Repeat by shorting 3.3 volts.	Y, ,Shorting either the 5 V or 3.3 V rail did not cause failure of the other rail. The unaffected rail remained stable and returned to normal once the short was removed.
No resets when motors run. With the cart driving, the ESP32 stays powered from this board (no brownouts/reboots) for 3 minutes.	Power the whole system, start/stop the motors several times, and drive in place for 3 min.	Y, ,During multiple cycles of starting and stopping the motors, the ESP32 and other subsystems remained powered with no brownouts or resets, confirming supply stability.
No big dip when something turns on. When the device is plugged in (drawing an extra 100 mA), the 5 V rail remains above 4.9 V, and the 3.3 V rail stays above 3.25 V.	With the board on, plug or unplug an extra 100 mA load a few times. Watch the multimeter.	Y, ,Plugging in or removing an additional 100 mA load caused only minimal voltage change. The 5 V and 3.3 V rails stayed within acceptable limits during transient events.

Table A.5 User Subsystem Requirements and Verification

Requirement	Verification	Verification status (Y or N)
Command work reliably indoors to at least 5 m.	The commands can be executed while standing 1, 3, 5, and 7 meters away from the cart. Pressing the mapped button checks if the command really works.	Partly Y, the signal is not stable at 5 m but works for about 4 m.
E-STOP is immediate	While driving at ~0.8 m/s, press E-STOP. Checking if the motors cut off and the cart stops.	Y, when sending stop command with BLE, stops immediately
No spurious action will be caused by noise or repeats	Hold a button for 3s. Checking if the cart performs one continuous action and no unintended mode changes occur.	Y, when there is no input with BLE, cart will not do action

Appendix B Code Excerpts

```
SpdPair controller(float dist_error, float steer_error, float d_ultrasound, float b_ultrasound) {
  float base = Kp * dist_error;
  float steer = Ki * steer_error;
  float l = base + steer;
  float r = base - steer;
  if (l < 0) l *= 1.5;
  if (r < 0) r *= 1.5;
  if (d_ultrasound >= 0 && d_ultrasound < 50){
    l = 0;
    r = 0;
  }
  if (b_ultrasound >= 0 && b_ultrasound < 60){
    l = 0;
    r = 0;
  }
  if (l > 255) l = 255;
  if (l < -255) l = -255;
  if (r > 255) r = 255;
  if (r < -255) r = -255;
  SpdPair s;
  if (stop_sign == true) {
    s.left = 0;
    s.right = 0;
  } else {
    s.left = (int)l;
    s.right = (int)r;
  }
  return s;
}
```

Figure B.1 Code for controller

```
struct SpdPair {
  int left;
  int right;
};


unsigned long startTime;

// === PWM setter ===
void motorSetDuty(int ch, uint8_t duty) {
  ledc_set_duty(LEDC_SPEED_MODE, (ledc_channel_t)ch, duty);
  ledc_update_duty(LEDC_SPEED_MODE, (ledc_channel_t)ch);
}

// === Motor Control ===
void setMotor(int in1, int in2, int ch, int speed) {
  if (speed > 0) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    motorSetDuty(ch, speed);
  } else if (speed < 0) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    motorSetDuty(ch, -speed);
  } else {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    motorSetDuty(ch, 0);
  }
}

void setMotorLeft(int speed) {
  setMotor(IN1_L, IN2_L, CH_L, speed);
}
void setMotorRight(int speed) {
  setMotor(IN1_R, IN2_R, CH_R, speed);
}
```

Figure B.2 Code for motor control function

```
// BLE controll
class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
    Serial.println("BLE device connected");
  }

  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    Serial.println("BLE device disconnected");
    BLEDevice::getAdvertising()->start();
  }
};

class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* pCharacteristic) {
    String rxValue = pCharacteristic->getValue();
    rxValue.trim();
    if (rxValue.length() == 0) return;

    Serial.print("BLE RX: ");
    Serial.println(rxValue);

    if (rxValue.startsWith("q")) {
      stop_sign = true;
    } else if (rxValue.startsWith("m")) {
      stop_sign = false;
    } else if (rxValue.startsWith("d")) {
      dist_error = rxValue.substring(1).toFloat();
    } else if (rxValue.startsWith("s")) {
      steer_error = rxValue.substring(1).toFloat();
    } else if (rxValue.startsWith("kp")) {
      Kp = rxValue.substring(2).toFloat();
    } else if (rxValue.startsWith("ki")) {
      Ki = rxValue.substring(2).toFloat();
    }
  }
};
```

Figure B.3 Functions for connecting and reading from APP based on BLE

```
void handleSerialInput() {
  while (Serial.available()) {
    char c = Serial.read();
    if (c == '\n' || c == '\r') {
      inputString.trim();
      if (inputString.startsWith("q")) {
        stop_sign = true;
      }
      //distance error d:
      if (inputString.startsWith("d:")) {
        dist_error = inputString.substring(2).toFloat();
        Serial.printf("New dist error = %.2f\n", dist_error);
      }
      //steering error s:
      else if (inputString.startsWith("s:")) {
        steer_error = inputString.substring(2).toFloat();
        Serial.printf("New steer error = %.2f\n", steer_error);
      }
      inputString = "";
    }
    else {
      inputString += c;
    }
  }
}
```

Figure B.4 Function for serial reading

```
// setup Serial port
Serial.begin(115200);
Serial.println("PWM dual motor test start...");
startTime = millis();
// --- Motor Setup ---
pinMode(STBY_PIN, OUTPUT);
digitalWrite(STBY_PIN, HIGH);
pinMode(TRIG_PIN_FRONT, OUTPUT);
pinMode(ECHO_PIN_FRONT, INPUT);
digitalWrite(TRIG_PIN_FRONT, LOW);
pinMode(TRIG_PIN_BACK, OUTPUT);
pinMode(ECHO_PIN_BACK, INPUT);
digitalWrite(TRIG_PIN_BACK, LOW);
pinMode(IN1_L, OUTPUT);
pinMode(IN2_L, OUTPUT);
pinMode(IN1_R, OUTPUT);
pinMode(IN2_R, OUTPUT);
ledc_timer_config_t timer_conf = {
    .speed_mode       = LEDC_SPEED_MODE,
    .duty_resolution  = LEDC_DUTY_BITS,
    .timer_num        = LEDC_TIMER,
    .freq_hz          = LEDC_BASE_FREQ,
    .clk_cfg          = LEDC_AUTO_CLK
};
ledc_timer_config(&timer_conf);
ledc_channel_config_t channel_conf_L = {
    .gpio_num       = PWM_L,
    .speed_mode     = LEDC_SPEED_MODE,
    .channel        = (ledc_channel_t)CH_L,
    .timer_sel      = LEDC_TIMER,
    .duty           = 0,
    .hpoint         = 0
};
ledc_channel_config(&channel_conf_L);
ledc_channel_config_t channel_conf_R = {
    .gpio_num       = PWM_R,
    .speed_mode     = LEDC_SPEED_MODE,
    .channel        = (ledc_channel_t)CH_R,
    .timer_sel      = LEDC_TIMER,
    .duty           = 0,
    .hpoint         = 0
};
ledc_channel_config(&channel_conf_R);
```

Figure B.5 Set up of serial port and motor driver

```
// BLE setup
BLEDevice::init("ESP32_CAR_BLE");
pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());
BLEService* pService = pServer->createService(SERVICE_UUID);
  // RX characteristic
pRxCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID_RX,
    BLECharacteristic::PROPERTY_WRITE
);
pRxCharacteristic->setCallbacks(new MyCallbacks());
// TX characteristic
pTxCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID_TX,
    BLECharacteristic::PROPERTY_NOTIFY
);
// BLE  CCCD descriptor
pTxCharacteristic->addDescriptor(new BLE2902());
pRxCharacteristic = pService->createCharacteristic(
  CHARACTERISTIC_UUID_RX,
  BLECharacteristic::PROPERTY_WRITE
);
pRxCharacteristic->setCallbacks(new MyCallbacks());
pService->start();
BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pAdvertising->start();
Serial.println("BLE ready, scan with phone.");
```

Figure B.6 Setup for BLE(bluetooth)

```
float readfrontDistanceCm() {
  digitalWrite(TRIG_PIN_FRONT, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG_PIN_FRONT, HIGH);
  delayMicroseconds(15);
  digitalWrite(TRIG_PIN_FRONT, LOW);

  unsigned long duration = pulseIn(ECHO_PIN_FRONT, HIGH, 60000); // over 60ms leads to timeout
  if (duration == 0) return -1;

  return duration * 0.0343 / 2.0;
}
float readbackDistanceCm() {
  digitalWrite(TRIG_PIN_BACK, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG_PIN_BACK, HIGH);
  delayMicroseconds(15);
  digitalWrite(TRIG_PIN_BACK, LOW);
  unsigned long back_duration = pulseIn(ECHO_PIN_BACK, HIGH, 60000); // over 60ms leads to timeout
  if (back_duration == 0) return -1;
  return back_duration * 0.0343 / 2.0;
}
```

Figure B.7 Function for reading data from Ultrasonic sensor

```
void loop() {
  handleSerialInput();
  float d_ultrasound = readfrontDistanceCm();
  float b_ultrasound = readbackDistanceCm();
  SpdPair spd = controller(dist_error, steer_error, d_ultrasound, b_ultrasound);
  // Write output on to APP through BLE
  if (deviceConnected) {
    char msg[64];
    snprintf(msg, sizeof(msg),
             "L=%d R=%d d=%.1fcm b=%.1fcm d_err=%.2f s_err=%.2f Kp=%.2f",
             spd.left, spd.right,d_ultrasound,b_ultrasound, dist_error, steer_error, Kp);
    pTxCharacteristic->setValue((uint8_t*)msg, strlen(msg));
    pTxCharacteristic->notify();
  }
  setMotorLeft(spd.left);
  setMotorRight(spd.right);
  delay(500);
}
```

Figure B.8 Main controller function logic