ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Automated Guided Vehicle Wireless Charging System with DSP-Based Position-Adaptive Frequency Control

---

Team #14

Jingzhou Ding (jd47@illinois.edu)
Jinru Cai (jinruc2@illinois.edu)
Jiaxin Cao (jiaxin18@illinois.edu)
Yaxin Li (yaxinl2@illinois.edu)

TA: Yuhang Wang

May 2026

Abstract

This report presents an automated guided vehicle (AGV) wireless charging system that combines vision-assisted docking with position-adaptive wireless power transfer (WPT). The vehicle uses a Raspberry Pi 4B for navigation, USB camera processing, and ArUco marker recognition, while an STM32 microcontroller performs low-level motor control. The charging station uses an ESP32 to communicate with the Raspberry Pi over Wi-Fi and Transmission Control Protocol (TCP), then forwards alignment data over a wired link to a TMS320F28335 digital signal processor (DSP). The DSP parses the corrected two-dimensional position data, applies a calibrated quadratic position-to-frequency model, and generates 62–67 kHz full-bridge pulse-width modulation (PWM) signals for the resonant power stage. The lithium battery charging target is 12.6 V at 2 A, or 25.2 W. A high-input trial near 15 V verified the available charging-power headroom but drew more than 3.5 A at the input, so routine testing was reduced to a 12 V input to protect battery lifetime, long-term reliability, and safety margin. Calibration data from 44 measured coil-offset points at 12 V input show optimal frequencies from 62.20 kHz to 66.42 kHz and measured output power from 15.70 W to 18.80 W.

Contents

1 Introduction

Wireless power transfer (WPT) is a useful charging method for electric vehicles and automated guided vehicles (AGVs) because it removes exposed plug contacts and allows charging to begin automatically when a vehicle docks. The main practical limitation is coil misalignment: when the transmitter (Tx) and receiver (Rx) coils are laterally offset, the magnetic coupling coefficient decreases and the converter efficiency drops [1], [2]. A previous generation of this project used a fixed-frequency charging system, so it could not compensate for the coupling changes that occur during realistic parking.

This project addresses that problem with two coordinated mechanisms. First, the AGV uses camera-based ArUco marker detection to estimate its pose relative to the charging station and reduce coil offset before charging. Second, the charging station uses DSP-based pulse frequency modulation (PFM) to adjust the CLLC resonant converter frequency after the remaining offset is known [3], [4]. The current implementation differs from the original design document by separating vehicle and station responsibilities more cleanly: the Raspberry Pi performs navigation and marker recognition on the vehicle, the STM32 performs vehicle motion control, and an ESP32 at the charging station bridges Wi-Fi/TCP data from the Raspberry Pi to the wired DSP control hardware.

1.1 Objective

The objective is to demonstrate an AGV that can drive to a charging station, align its onboard receiver coil with the station transmitter coil, and receive wireless charging power with frequency compensation for residual misalignment. The intended result is a safer and more autonomous charging process than plug-in charging, with better efficiency than a fixed-frequency WPT system under imperfect parking.

1.2 High-Level Requirements

1. **Wireless charging performance:** The battery-rated charging target is 12.6 V at 2 A, corresponding to 25.2 W. A high-input test near 15 V confirmed that the power stage has the headroom needed for the rated charging point, but the input current exceeded 3.5 A. For battery lifetime, long-term reliability, and safety, routine testing is therefore performed at 12 V input. DC-to-DC transfer efficiency shall remain at least 75% for lateral coil misalignment up to ± 2 cm when adaptive frequency control is active.
2. **Vision and parking accuracy:** The Raspberry Pi and camera shall estimate ArUco marker position with ± 5 mm lateral accuracy and $\pm 3^\circ$ angular accuracy at the operating distance. After precision parking, the remaining coil misalignment shall be no more than 1 cm.
3. **Adaptive frequency control:** The DSP shall adjust the CLLC switching frequency over 62–67 kHz based on received offset data. After the Raspberry Pi finishes its

averaged final alignment measurement, the packet-transfer and DSP-update latency shall be no more than 50 ms.

4. **Output voltage stability:** The receiver output voltage ripple shall be no more than 0.5 V peak-to-peak during steady-state charging.

1.3 Block Diagram

Figure 1 shows the updated system architecture. Unlike the original design document, the Raspberry Pi no longer sends station-side frequency-control data directly to the DSP. Instead, the station ESP32 receives alignment data over Wi-Fi/TCP and forwards it to the DSP through a wired station-side link.

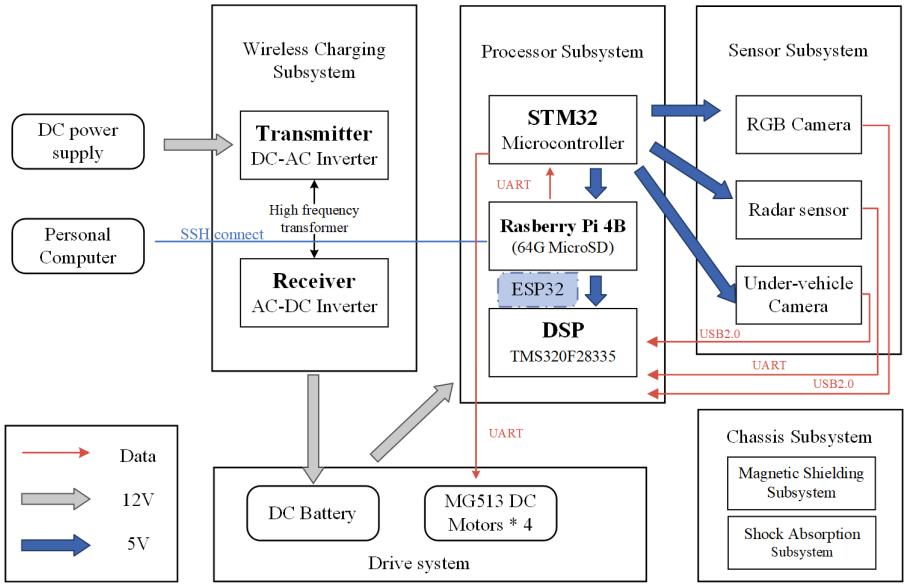

Figure 1: Overall AGV wireless charging system architecture, including the wireless charging, processor, sensor, drive, and chassis subsystems.

1.4 Project Changes

The main architectural change from the design document is the addition of an ESP32 at the charging station. The Raspberry Pi remains on the AGV and handles navigation, camera input, and ArUco marker recognition. The STM32 remains on the AGV and focuses on low-level motion control. The station-side ESP32 now handles the wireless TCP connection to the Raspberry Pi and relays alignment data to the DSP over a wired link, which reduces the need for the DSP to manage networking and keeps the high-frequency power stage control localized at the charging station.

2 Design

2.1 System Architecture

The final design is divided into two physical platforms. The AGV carries the Raspberry Pi, camera, STM32, motor driver, encoders, receiver coil, rectifier, and battery/load interface. The charging station contains the ESP32, DSP, gate drivers, full-bridge inverter, transmitter coil, and adjustable DC input supply. This split keeps navigation and motion control on the moving vehicle while keeping power conversion and frequency control at the fixed charging station.

The data path begins when the Raspberry Pi detects the ArUco marker mounted at the charging station. The Raspberry Pi estimates the lateral offset Δx, longitudinal offset Δy, and yaw error $\Delta\theta$ between the AGV receiver coil and the station transmitter coil [5], [6]. It then sends motion commands to the STM32 for precision parking and sends the final alignment data to the station ESP32 over a TCP socket. The ESP32 relays the received packet over a wired station-side link to the DSP. The DSP uses this offset to select a switching frequency for the CLLC converter.

2.2 Design Alternatives and Corrective Actions

Several design decisions changed during the semester as the team moved from the proposal architecture to the integrated prototype. Tables 1 and 2 summarize the main alternatives, the selected approach, and the corrective actions taken when an early approach was not robust enough for the final system. The main design pattern was to keep high-bandwidth sensing and navigation on the Raspberry Pi, time-critical actuation on the STM32, network handling on the ESP32, and switching control on the DSP.

2.3 Wireless Charging Subsystem

The wireless charging subsystem transfers power across an air gap to the AGV battery and load interface. The power stage uses a CLLC resonant converter because its gain changes with switching frequency [3]. This allows the controller to compensate for coupling changes caused by coil misalignment. The implemented controller uses 64 kHz as the baseline frequency and limits adaptive operation to 62–67 kHz.

Figure 2 shows the charging-specific power and data paths. The Raspberry Pi on the intelligent car provides the final position information to the station ESP32 over Wi-Fi/TCP, and the ESP32 forwards the position information to the DSP. The DSP then controls the transmitter full bridge. Energy flows from the transmitter full bridge through the transmitter coil, across the wireless coupling gap, through the receiver coil and receiver full bridge, and finally to the vehicle battery.

Table 3 summarizes the measured coil and compensation-component values. The transmitter and receiver coils have similar inductance, with $L_p = 168.34\ \mu$H on the transmitter side and $L_s = 159.75\ \mu$H on the receiver side, which is consistent with an approximately

Table 1: Power and control design alternatives.

Design Issue	Alternatives Considered	Selected Approach	Justification or Corrective Action
WPT frequency control	Fixed 64 kHz operation; manual sweep; position-adaptive PFM.	DSP-based position-adaptive PFM over 62–67 kHz.	Fixed-frequency operation could not compensate for coupling changes. The calibrated model has a mean absolute frequency error of 0.10 kHz on the 44 measured points.
Position-to-frequency mapping	Lookup table interpolation; one-dimensional fit; two-dimensional quadratic fit.	Two-dimensional quadratic model using corrected x and y offsets.	The measured calibration data vary in both directions, so a two-dimensional model gives smoother updates than a sparse lookup table. Output is clamped to 62–67 kHz for protection.
Vehicle actuation	Raspberry Pi directly commands motor PWM; STM32 local motor loop.	Raspberry Pi sends velocity commands; STM32 closes the motor-speed loop.	The STM32 updates encoder-based PI control every 10 ms, while the Raspberry Pi remains responsible for navigation-level decisions.
Charging test input	Routine operation near 15 V input; reduced 12 V input for calibration.	12 V routine testing with one higher-input headroom trial.	The near-15 V trial showed rated-power headroom, but input current exceeded 3.5 A. The team returned to 12 V input to reduce thermal stress and battery-life risk while the draft verification remains incomplete.

Table 2: System integration and sensing design alternatives.

Design Issue	Alternatives Considered	Selected Approach	Justification or Corrective Action
Controller partitioning	Raspberry Pi directly drives DSP data path; DSP handles communication; ESP32 station bridge.	ESP32 receives Raspberry Pi TCP packets and forwards raw bytes to the DSP.	The bridge keeps the Wi-Fi/TCP stack off the DSP and keeps high-frequency PWM control local to the charging station.
Wireless protocol	UDP datagrams; TCP socket connection.	TCP connection from ESP32 client to Raspberry Pi server.	The data rate is low, but packet order and delivery matter for safe frequency updates. TCP reliability is more useful than UDP's lower overhead.
Vision pose estimation	Full PnP pose solve; calibrated image-plane offset.	Calibrated image-plane offset with yaw from the marker top edge.	The docking geometry is fixed, so the simpler method reduces processing complexity. Measured zero offsets of $x_0 = 5.01$ mm and $y_0 = 49.38$ mm are subtracted before transmission.

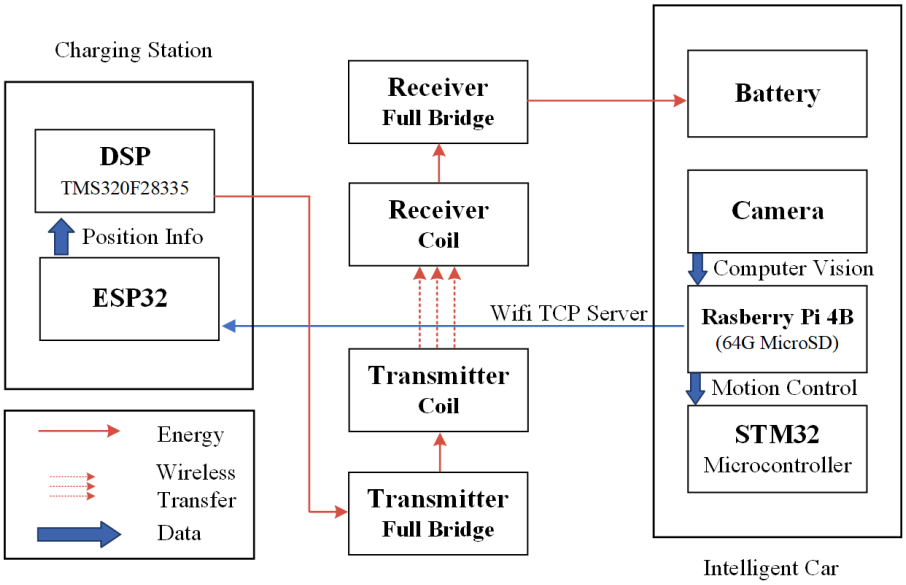

Figure 2: Wireless charging subsystem architecture with station-side DSP control, ESP32 communication, coil-to-coil wireless transfer, and vehicle-side battery charging path.

1:1 coil turns ratio. The measured series-aiding and series-opposing inductances were $L_+ = 596.4\ \mu\text{H}$ and $L_- = 57.4\ \mu\text{H}$. For coupled coils,

$$M = \frac{L_+ - L_-}{4}, \qquad k = \frac{M}{\sqrt{L_p L_s}}, \tag{1}$$

which gives a measured mutual inductance of $M = 134.75\ \mu\text{H}$ and coupling coefficient $k = 0.822$ at the tested alignment. The average value $(L_+ + L_-)/2 = 326.9\ \mu\text{H}$ is close to the separately measured sum $L_p + L_s = 328.09\ \mu\text{H}$, which supports the consistency of the coupled-inductor measurement.

Using the measured coil inductances and compensation capacitors, the simple single-side LC estimates are

$$f_p = \frac{1}{2\pi\sqrt{L_p C_p}} = 85.26\ \text{kHz}, \qquad f_s = \frac{1}{2\pi\sqrt{L_s C_s}} = 85.09\ \text{kHz}. \tag{2}$$

These values show that the transmitter and receiver compensation networks are closely matched. However, the single-side LC resonance does not directly determine the final operating frequency because the two resonators are strongly coupled. With a high measured coupling coefficient, the coupled system exhibits frequency splitting: the original single-resonator peak separates into a lower-frequency mode and a higher-frequency mode. For two closely matched resonators, the approximate split frequencies can be interpreted as

Table 3: Measured coil and compensation parameters.

Parameter	Value	Notes
Transmitter coil inductance, L_p	168.34 μH	Charging-station Tx coil.
Receiver coil inductance, L_s	159.75 μH	AGV Rx coil.
Series-aiding inductance, L_+	596.4 μH	Used to estimate mutual inductance.
Series-opposing inductance, L_-	57.4 μH	Used to estimate mutual inductance.
Tx compensation capacitor, C_p	20.7 nF	Measured transmitter-side capacitor.
Rx compensation capacitor, C_s	21.9 nF	Measured receiver-side capacitor.
Estimated mutual inductance, M	134.75 μH	$(L_+ - L_-)/4$.
Estimated coupling coefficient, k	0.822	At the tested coil alignment.

$$f_{\text{low}} \approx \frac{f_0}{\sqrt{1+k}}, \qquad f_{\text{high}} \approx \frac{f_0}{\sqrt{1-k}}. \tag{3}$$

Using $f_0 \approx 85.2$ kHz and $k = 0.822$, the lower split mode is approximately 63 kHz, which is consistent with the experimentally selected DSP operating range of 62–67 kHz. Therefore, the 85 kHz value in Equation (2) describes the isolated component-level LC tanks, while the adaptive control range follows the lower resonance peak of the loaded, strongly coupled WPT system.

The lithium battery rated charging point is 12.6 V at 2 A, so the rated charging power is

$$P_{\text{rated}} = 12.6\ \text{V} \times 2\ \text{A} = 25.2\ \text{W}. \tag{4}$$

During the reported calibration measurements, the power-stage input was held at 12 V to reduce thermal and electrical risk while validating the position-to-frequency behavior. The measured output power therefore remained below the full battery-rated charging power. The team also performed a higher-input test near 15 V, which confirmed the available headroom for the rated charging point. However, the input current exceeded 3.5 A at this operating condition, increasing thermal stress and battery-life risk. For long-term operation and safety margin, the system was returned to 12 V input for routine testing and data collection.

The DSP generates PWM signals for the gate drivers and updates the timer period when a new frequency command is selected. If f_{clk} is the DSP timer clock and f_{sw} is the target switching frequency, the timer period is set according to

$$\text{TBPRD} = \frac{f_{\text{clk}}}{f_{\text{sw}}} - 1. \tag{5}$$

The DSP should disable PWM output during faults such as overtemperature, invalid alignment packets, or output overcurrent.

2.4 Vision and Navigation Subsystem

The Raspberry Pi performs the high-level vehicle functions. It receives frames from a USB camera, detects ArUco marker corners using OpenCV, computes the relative coil offset, and decides when to transition from route following to marker-guided docking. In the updated architecture, this same Raspberry Pi also owns navigation decisions and sends motion commands to the STM32 during precision parking.

The implemented image-recognition node runs as the ROS node `aruco_docking_node`. It subscribes to the Boolean topic `/start_detect`; when the docking manager publishes `True`, the node opens camera 0 at 1280×720 resolution and searches for a marker from OpenCV's 4-by-4, 50-marker ArUco dictionary. The code supports both newer and legacy OpenCV ArUco detector APIs, which makes the same script usable across the OpenCV versions installed on the Raspberry Pi.

Rather than solving a full camera pose with PnP, the current implementation uses a calibrated image-plane offset. For the first detected marker, the Raspberry Pi averages the four marker-corner coordinates to obtain the marker center. It subtracts the image center to compute `dx_pixel` and `dy_pixel`, then multiplies by the measured scale factor 0.035 cm/pixel. The marker yaw estimate is computed from the angle of the top marker edge. Before transmission, the centimeter offsets are converted to millimeters and corrected by the measured visual zero offset, $x_0 = 5.01$ mm and $y_0 = 49.38$ mm:

$$x_{\text{corr}} = 10\Delta x_{\text{cm}} - 5.01, \qquad y_{\text{corr}} = 10\Delta y_{\text{cm}} - 49.38. \tag{6}$$

To reduce frame-to-frame noise, one detection trigger performs five independent samples. Each sample lasts up to 1.2 s and is accepted once at least eight valid marker frames have been collected. The node then averages the accepted sample means and publishes the corrected result on `/aruco_offset` as a `Vector3`, where x and y are corrected millimeter offsets and z is the marker angle in degrees. It also publishes text state messages such as `DETECTING`, `DONE`, `NO_MARKER`, and `ERROR` on `/aruco_status`. If no valid marker is found, the Raspberry Pi sends a marker-lost status frame instead of an OK frame.

The camera is mounted on the AGV and directed toward the charging station marker. No image frames need to leave the Raspberry Pi during normal operation. Only compact

numeric data such as Δx, Δy, $\Delta\theta$, packet sequence number, and validity status are transmitted to other controllers. The node also writes a small text log and one debug image on the Raspberry Pi for field debugging when ROS console output is delayed.

2.5 Motion Control Subsystem

The STM32 microcontroller performs the low-level vehicle motion control [7]. The Raspberry Pi remains responsible for navigation and docking decisions, while the STM32 handles time-critical motor actuation, encoder feedback, PWM generation, and local safety behavior. During approach and precision docking, the Raspberry Pi publishes velocity commands on the ROS `/cmd_vel` topic. The ROS base node converts these commands to an 11-byte serial frame and sends them to the STM32 at 115200 baud.

The command frame contains the desired chassis velocity in `linear.x`, `linear.y`, and `angular.z`. The values are scaled by 1000 before transmission, giving 1 mm/s linear-speed resolution and 0.001 rad/s yaw-rate resolution. For the differential-drive docking motion used in this project, `linear.y` is kept at zero; forward and backward corrections use `linear.x`, while heading corrections use `angular.z`. The STM32 parses the command, converts it to left- and right-wheel targets, and applies the appropriate motor polarity and PWM output.

Wheel speed is regulated locally on the STM32 using encoder feedback. The firmware reads the encoder timer counts every 10 ms and updates an incremental PI speed controller,

$$PWM[k] = PWM[k-1] + K_P(e[k] - e[k-1]) + K_I e[k], \tag{7}$$

where $e[k]$ is the difference between measured and target wheel speed. The calculated PWM command is clamped before being written to the motor driver; the main control path limits the motor command to 6900 counts, and the PI controller includes a hard limit of 7200 counts. The STM32 also sends a 24-byte feedback frame to the Raspberry Pi containing chassis velocity, IMU data, battery voltage, frame markers, and an XOR checksum.

The motion-control path includes basic fail-safe behavior. Frames use fixed markers and an XOR checksum, so malformed packets are rejected instead of being interpreted as valid velocity commands. When the ROS base node shuts down, it sends a zero-velocity command before closing the serial port. The STM32 firmware also sets the motor PWM to zero under undervoltage conditions, and the docking logic publishes zero velocity when marker detection is lost or final parking tolerance is reached.

2.6 Charging Station Communication and DSP Control

The ESP32 is the network bridge at the charging station. It connects to the Raspberry Pi through Wi-Fi and TCP, with the ESP32 acting as a TCP client for the Raspberry Pi server at `192.168.0.100:5000`. The Raspberry Pi script binds the server socket to `0.0.0.0:5000`,

Table 4: Key motion-control implementation parameters.

Parameter	Value	Purpose
Serial baud rate	115200 baud	Raspberry Pi–STM32 communication.
Command frame length	11 bytes	Velocity command from ROS to STM32.
Feedback frame length	24 bytes	Odometry, IMU, and battery feedback.
Velocity scale factor	$1000\times$	m/s to mm/s; rad/s to mrad/s.
Motor control period	10 ms	Encoder read and PI speed update.
Command wire time	0.95 ms	$11 \times 10/115200$.
Feedback wire time	2.08 ms	$24 \times 10/115200$.
Main PWM clamp	6900 counts	Prevents excessive motor command.
Hard PI clamp	7200 counts	Prevents PI windup and saturation.

accepts ESP32 connections in a background thread, and replaces the old socket whenever a new ESP32 connection arrives. The ESP32 receives the Raspberry Pi data stream and transparently forwards the raw bytes to UART2 without modifying or rebuilding the frame. UART2 is configured as 115200 baud, 8N1, with GPIO17 as TX and GPIO16 as RX. This design keeps the TCP stack off the DSP and allows the DSP firmware to focus on deterministic PWM and protection logic.

TCP was selected instead of UDP because the transmitted alignment packets contain low-bandwidth but safety-critical control information. These packets include corrected marker offsets, packet sequence number, and status flags required by the downstream DSP frequency-control subsystem. Packet loss or out-of-order delivery could lead to incorrect resonant-frequency updates or invalid docking behavior. TCP provides reliable, ordered delivery, automatic retransmission, and built-in connection management while adding negligible bandwidth overhead for the compact packets used in this project. Since the system sends numerical alignment results rather than image streams, the TCP latency overhead is small compared with the overall docking cycle.

Another advantage of the TCP client-server structure is that the Raspberry Pi only needs to maintain a single persistent socket after the ESP32 client connects. Once connected, the Raspberry Pi can transmit docking packets through the same socket without tracking dynamically changing ESP32 source ports, which simplifies the communication software and improves robustness during long-duration experiments. Table 5 summarizes the protocol tradeoff.

Table 5: Comparison between TCP and UDP for docking communication.

Feature	TCP	UDP
Reliable delivery	Yes	No
Ordered packets	Yes	No
Connection management	Yes	No
Packet retransmission	Yes	No
Docking-control data	Suitable	Limited
High-bandwidth streaming	Moderate	Excellent

Table 6 lists the 13-byte position frame sent by the Raspberry Pi and forwarded by the ESP32. The ESP32 code also parses the frame header, floating-point offsets, and XOR checksum for serial-monitor debugging; however, the control path remains transparent raw-byte forwarding from TCP to UART2. The firmware includes Wi-Fi reconnect, TCP reconnect, received-byte statistics, and LED connection-state indication. After a TCP disconnect, the ESP32 retries the Raspberry Pi server approximately every 2 s.

On the Raspberry Pi side, each result is sent as a short burst instead of a single packet. The script transmits the same final status and corrected offset at 30 Hz for 1 s while incrementing an 8-bit sequence number. This burst behavior was chosen because the ESP32-to-DSP UART path is one-way and does not provide an acknowledgement. The Raspberry Pi currently sends `STATUS=0` for a valid marker result and `STATUS=1` for marker lost; `STATUS=2` is reserved for link-down handling in the broader protocol.

Table 6: Position frame sent from Raspberry Pi to ESP32 and forwarded to the DSP.

Field	Format	Purpose
Header	`AA 55`	Two-byte frame marker.
SEQ	uint8	Sequence number for detecting dropped or repeated packets.
STATUS	uint8	0 = OK, 1 = marker lost, 2 = link down.
X	float32 little-endian	Corrected x offset in millimeters.
Y	float32 little-endian	Corrected y offset in millimeters.
XOR	uint8	XOR checksum of SEQ, STATUS, X, and Y body bytes.

On the DSP side, the F28335 implementation is based on TI's ePWM up-down action-qualifier example. The DSP receives the ESP32-forwarded position frame through SCI-C at 115200 baud. A byte-wise receive interrupt implements a state-machine parser for the `AA 55` header, sequence number, status byte, little-endian floating-point `x_mm` and `y_mm`

fields, and XOR checksum. When the checksum passes and the status byte is OK, the firmware updates the latest valid position sample.

The latest firmware does not use a simple lookup table. Instead, the DSP applies the two-dimensional quadratic model in Equation (8), which was fitted from measured coil-offset calibration data:

$$\begin{aligned} f_{\mathrm{kHz}} = {} & 62.2614205 - 0.00922142x + 0.00947202y \\ & + 0.000394587x^2 + 0.0000559534xy + 0.000355344y^2, \end{aligned} \tag{8}$$

where x and y are corrected position offsets in millimeters. Before applying the model, the DSP checks that each coordinate is within ± 110 mm and that the radial offset $\sqrt{x^2 + y^2}$ is no more than 115 mm. The computed frequency is then clamped to 62–67 kHz. If the position is invalid or stale, the controller returns to the 64 kHz baseline frequency.

The DSP configures ePWM1 and ePWM2 for full-bridge drive using up-down counting, 50% duty cycle, synchronized phase-shift control, and 500 ns dead time. ePWM1 is the reference bridge leg and ePWM2 is shifted by 180 degrees through the phase register. The commanded frequency is not applied as a step; instead, a 10 ms control loop applies a slope limit so that the power stage tracks the new target gradually.

Fault handling is implemented through communication watchdogs, stale-result timeout, range checks, ePWM Trip Zone behavior, and a GPIO12-driven disable signal. During abnormal conditions, the DSP disables power output or returns to the 64 kHz baseline. The firmware also preserves CCS Watch manual-frequency mode and sweep mode for resonant-point debugging, power-stage response testing, and future recalibration.

3 Verification

3.1 Verification Procedure

Verification should demonstrate that the final system works as an integrated AGV charging platform rather than as isolated modules. The most important tests are charging power and efficiency, vision accuracy, docking accuracy, TCP-to-DSP communication latency, and output voltage stability.

3.2 Results

Table 7 summarizes the top-level verification plan. The result column includes the coil-offset calibration data that are currently available from Sheet2 of the measurement workbook; full-system charging and docking values should be replaced with final demonstration data after the final tests.

3.3 Discussion

The motion-control verification is included because the STM32 is the actuator layer that determines whether the visual alignment result becomes repeatable physical docking motion. The basic command-path test starts the base serial node, sends low-speed `/cmd_vel` commands, verifies wheel response, sends a zero-velocity command before disabling the vehicle, and checks that malformed serial frames are rejected. This command path is verified for the current draft. Additional final docking tests should still check small yaw-rate corrections and safe stopping when marker detection or valid command input is lost.

The vision test should match the implemented Raspberry Pi behavior. A test fixture can place the marker at known image-plane offsets and angles, then trigger the ROS topic `/start_detect`. Each trigger should produce five sample windows of up to 1.2 s, with at least eight valid marker frames per accepted sample. The reported x and y values are checked after the zero-offset correction in Equation (6). The measured position offset error is no more than 0.5 mm, which satisfies the ± 5 mm position requirement. Tests should also cover the no-marker case and confirm that the Raspberry Pi sends the marker-lost status rather than a stale OK offset.

The latency requirement applies to the communication and DSP update path after the Raspberry Pi has produced its averaged alignment result. The recognition routine intentionally takes multiple samples for a steadier final docking measurement, so the complete trigger-to-result time is longer than the packet-transfer latency. For the communication test, the expected Raspberry Pi behavior is a 30 Hz, 1 s burst of 13-byte frames with monotonically increasing 8-bit sequence numbers. The final-result transfer latency is verified to meet the 50 ms requirement.

The communication verification is different from the original design document because the frequency-control path now crosses a Wi-Fi/TCP link before reaching the station-

Table 7: Verification summary for the updated architecture.

Requirement	Test Method	Result
Support the 12.6 V, 2 A battery-rated charging point	Connect a load or battery interface; measure output voltage and current at 12 V input and during the high-input trial.	12 V calibration: 15.70–18.80 W. Near-15 V trial showed power headroom but input current exceeded 3.5 A.
Maintain at least 75% efficiency under ± 2 cm offset	Measure input and output power at known offsets with adaptive control enabled.	16–17 W output: about 60%. Resonant point: 1–2 W output with $>90\%$.
Vision accuracy within ± 5 mm and $\pm 3^\circ$	Compare Raspberry Pi offset and angle estimates with fixture measurements.	Offset error ≤ 0.5 mm.
Residual parking error no more than 1 cm	Run docking trials and measure final Tx–Rx coil offset.	TBD
Motion-control command path is safe and repeatable	Send ROS `/cmd_vel` commands through the base node; verify wheel response, stopping behavior, and malformed-frame rejection.	Verified.
Final-result transfer latency no more than 50 ms	Timestamp Raspberry Pi result frame, ESP32 TCP receipt, wired forwarding, and DSP PWM update.	Verified.
Output ripple no more than 0.5 V peak-to-peak	Measure receiver output with an oscilloscope during charging.	1.0 Vpp under electronic-load CV mode; not met.

Table 8: ROS commands used for motion-control bring-up.

Step	Command	Purpose
Start base	`roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch car_mode:=mini_diff`	Opens the STM32 serial base node.
Move slowly	`rostopic pub -r 10 /cmd_vel geometry_msgs/Twist linear.x=0.05, angular.z=0.0`	Verifies low-speed wheel response.
Stop	`rostopic pub -1 /cmd_vel geometry_msgs/Twist linear.x=0.0, angular.z=0.0`	Sends a final zero-velocity command.

Table 9: Additional motion-control verification items.

Requirement	Verification method	Result
STM32 accepts valid docking commands and rejects malformed serial data.	Send valid `/cmd_vel` commands and intentionally corrupted serial frames; verify that valid frames move the wheels and corrupted frames do not update motor PWM.	Verified.
Motor response is repeatable for low-speed docking.	Command $v = 0.05$ m/s and $v = -0.05$ m/s for fixed time windows; measure distance traveled and final stopping error over at least five trials.	Verified for command response.
Yaw correction is controllable near the charger.	Command small positive and negative ω_z values; measure heading change and confirm that the vehicle can reduce ArUco yaw error without overshoot.	To be measured.
Loss of valid command stops the vehicle safely.	Stop the docking node or publish a marker-lost state; verify that the Raspberry Pi sends zero velocity and that motor PWM returns to zero.	To be measured.

side ESP32 and DSP. Therefore, final testing must include packet loss, reconnect behavior, stale-packet rejection, and DSP behavior when no valid alignment data are available. A conservative fail-safe behavior is to stop adaptive updates or disable charging until a valid packet is received.

The available coil calibration data contain 44 measured offset points. The measured offsets span $x = -75.81$ mm to 93.40 mm and $y = -87.40$ mm to 61.40 mm. The measured optimal switching frequencies span 62.20–66.42 kHz, and the measured output powers span 15.70–18.80 W with an average of 16.68 W under the 12 V input calibration setup. These data do not represent the maximum battery charging capability; they were collected under the reduced input voltage selected for routine testing. The efficiency measurement showed a tradeoff between delivered power and transfer efficiency: when the system was tuned to maintain about 16–17 W output power, the measured efficiency was about 60%; when the frequency was adjusted back to the design resonance point, the efficiency increased to above 90%, but output power dropped to about 1–2 W. Therefore, the 75% efficiency requirement was not simultaneously met with the higher-power operating point in the present setup. The battery-rated charging point is 25.2 W. A high-input trial near 15 V showed that the system has the necessary charging-power headroom, but the input current exceeded 3.5 A. The team therefore returned to 12 V input to reduce battery aging, thermal loading, and long-term safety risk. Under electronic-load constant-voltage mode, the measured receiver ripple is 1.0 Vpp, so the 0.5 Vpp ripple requirement is not yet met. When Equation (8) is evaluated at the measured offsets, its frequency error relative to the Sheet2 optimal frequencies has a mean absolute error of 0.10 kHz and a root-mean-square error of 0.13 kHz. The complete calibration table is included in Appendix B.

4 Costs

4.1 Parts and Labor Costs

The project reuses several major components from the previous generation or existing lab inventory, including the AGV chassis, motors, encoders, H-bridge motor driver, STM32 development board, TMS320F28335 DSP board, ESP32 module, and 12 V AGV battery. The two Raspberry Pi boards were newly purchased for this project rather than inherited from the previous design. Table 10 lists both the estimated retail value and the amount paid by the team or department. Lab-owned and reused parts are listed with a paid cost of \$0.00 so that the implementation cost and replacement value are both visible.

The reimbursement document lists parts purchases in RMB. These values were converted to U.S. dollars using an exchange rate of

$$1 \text{ USD} = 6.803 \text{ RMB}. \tag{9}$$

Thus, the parts-cost conversion is

$$C_{\text{USD}} = \frac{C_{\text{RMB}}}{6.803}. \tag{10}$$

The ECE 445 guideline recommends estimating labor cost for each partner as

$$\text{labor cost} = \text{ideal hourly salary} \times \text{actual hours spent} \times 2.5. \tag{11}$$

For this draft estimate, Table 11 uses an ideal engineering salary of \$38.00/h and 50 h per partner. These hours should be replaced with the final individual time records before submission if the team keeps a more detailed log.

Table 12 summarizes the resulting project cost. The paid total is the amount paid by the team or department, while the retail-equivalent total also counts reused lab-owned parts at their estimated replacement value.

No machine shop cost is expected. If a charging station enclosure is needed, it can be 3D printed through department resources. The current prototype is not intended for immediate mass production, so the cost estimate does not include a bulk-purchase or manufacturing-volume projection.

4.2 Schedule

Table 13 summarizes the semester schedule by week and team member. The schedule is written as a draft record of the work distribution used for the current report version; it should be updated if the final experiment dates or individual responsibilities change before submission.

Table 10: Draft parts cost with retail value and amount paid.

Item	Retail Cost (USD)	Paid Cost (USD)	Source or Status
Power-electronics components, connectors, and shipping from LCSC receipts	$94.59	$94.59	Purchased
Transmitter PCB fabrication	$7.44	$7.44	Purchased
Receiver PCB fabrication	$4.76	$4.76	Purchased
Raspberry Pi 4B, 8 GB development board	$126.56	$126.56	Purchased
Raspberry Pi 3B kit for development and backup testing	$53.81	$53.81	Purchased
USB-to-TTL serial adapter	$1.43	$1.43	Purchased
Raspberry Pi USB camera	$2.67	$2.67	Purchased
USB HD camera module	$13.23	$13.23	Purchased
AGV chassis, motors, encoders, H-bridge motor driver, STM32 board, and 12 V battery	$300.00	$0.00	Reused lab inventory
TMS320F28335 DSP board	$180.00	$0.00	Reused lab inventory
ESP32 station communication module	$8.00	$0.00	Reused lab inventory
Wiring, headers, fasteners, and mechanical mounting hardware	$20.00	$0.00	Lab stock
Parts subtotal	$812.50	$304.50	

Table 11: Draft labor cost estimate by team member.

Team Member	Hourly Rate	Hours	Labor Cost
Jingzhou Ding	$38.00/h	50	$4,750.00
Jinru Cai	$38.00/h	50	$4,750.00
Jiaxin Cao	$38.00/h	50	$4,750.00
Yaxin Li	$38.00/h	50	$4,750.00
Labor subtotal		200	$19,000.00

Table 12: Draft project cost summary.

Category	Paid Cost	Retail-Equivalent Cost
Parts subtotal	\$304.50	\$812.50
Labor subtotal	\$19,000.00	\$19,000.00
Grand total	\$19,304.50	\$19,812.50

Table 13: Draft weekly project schedule by team member.

Week	Jingzhou Ding	Jinru Cai	Jiaxin Cao	Yaxin Li
1	Reviewed DSP and WPT goals.	Audited coils and power hardware.	Audited AGV motion hardware.	Reviewed navigation requirements.
2	Defined adaptive-frequency control path.	Selected compensation and PCB direction.	Reviewed STM32 serial and motor code.	Defined Raspberry Pi software tasks.
3	Set up DSP ePWM baseline.	Prepared power-stage component list.	Tested low-speed motor commands.	Brought up camera and ArUco detection.
4	Defined packet fields for DSP parser.	Laid out transmitter and receiver boards.	Checked encoder feedback path.	Built ROS detection trigger flow.
5	Implemented SCI receive parser.	Assembled and inspected PCBs.	Tuned PI speed-control behavior.	Calibrated image-plane offset scale.
6	Added PWM update and clamp logic.	Brought up low-input WPT test.	Verified serial frame checks.	Prototyped TCP server on Raspberry Pi.
7	Added stale-packet fallback.	Measured coil inductance and capacitors.	Tested docking-speed limits.	Integrated ESP32 bridge link.
8	Tested manual frequency sweep.	Collected preliminary power data.	Integrated marker-lost stop behavior.	Added result burst transmission.
9	Fit quadratic frequency model.	Collected coil-offset calibration data.	Supported AGV docking trials.	Logged corrected marker outputs.
10	Integrated model into DSP firmware.	Compared 12 V and near-15 V input tests.	Checked command shutdown behavior.	Tested TCP reconnect behavior.
11	Verified frequency clamp and slope limit.	Measured output power and efficiency tradeoff.	Prepared repeatability test method.	Prepared latency test method.
12	Integrated station-side controller.	Prepared ripple and thermal tests.	Integrated AGV parking sequence.	Integrated vision output with packet sender.
13	Analyzed frequency-fit error.	Documented measured power-stage data.	Documented motion-control parameters.	Documented communication protocol.
14	Drafted DSP and verification sections.	Drafted WPT and cost data.	Drafted motion and safety material.	Drafted vision and communication sections.
15	Prepared final DSP test updates.	Prepared final charging test updates.	Prepared final docking test updates.	Prepared final vision test updates.

5 Conclusions

The updated prototype separates the system into vehicle-side autonomy and station-side charging control. The Raspberry Pi handles navigation and ArUco recognition, the STM32 handles motor control, the ESP32 bridges Wi-Fi/TCP data at the station, and the DSP controls the CLLC converter frequency.

At the current draft stage, the strongest verified result is the position-adaptive frequency-control path. The WPT calibration data contain 44 measured coil-offset points at 12 V input. The measured optimal switching frequency spans 62.20–66.42 kHz, the output power spans 15.70–18.80 W, and the fitted two-dimensional frequency model has a 0.10 kHz mean absolute error. A higher-input trial near 15 V showed headroom toward the 25.2 W battery-rated charging target, but input current exceeded 3.5 A. The 75% efficiency requirement has not yet been verified at the higher-power operating point: 16–17 W operation produced about 60% efficiency, while the high-efficiency resonant point delivered only 1–2 W.

5.1 Remaining Uncertainties

Remaining work includes final docking error, station forwarding, marker-loss, and fault tests. It should also reduce the measured 1.0 Vpp receiver ripple in electronic-load CV mode. If rated-operation targets remain unmet, likely fixes include retuning the network, reducing coil spacing, improving parking, revising the operating range, or separating the power and efficiency requirements.

5.2 Future Work

Future work should first complete the remaining verification tests with quantitative pass or fail data. A useful hardware improvement is a small data screen for live charging status. The PCB already reserves a sampling-chip interface, so a future revision can sample voltage and current, compute $P = VI$, and display voltage, current, power, frequency command, alignment status, fault state, and real-time efficiency. The same feedback path could later support current limiting, overcurrent shutdown, and automatic derating.

5.3 Ethics and Safety

This project follows the IEEE Code of Ethics by prioritizing safety, truthful reporting, and responsible use of autonomous and wireless power technology [8]. This draft reports unmet efficiency and incomplete verification items rather than presenting the prototype as fully validated. Personnel should stay away from active coils, field exposure near 64 kHz should be checked against relevant guidance when possible [9], and the transmitter should disable PWM for invalid, stale, or out-of-range packets. Broader impacts include reduced connector wear, operator effort, and wasted energy.

Appendix A Requirement and Verification Table

Table 14 expands the main verification summary with concrete pass criteria. Results marked TBD should be completed after final testing.

Table 14: Detailed requirement and verification table.

Requirement	Verification Method	Result
Battery-rated charging point is 12.6 V at 2 A.	Measure voltage and current; compute output power at 12 V input and during high-input trial.	Verified
Higher input should remain safe for long-term operation.	Raise input from 12 V to about 15 V and monitor input current and thermal behavior.	$>$3.5 A; 12 V used
Efficiency is at least 75% at offsets up to $\pm$2 cm.	Set known coil offsets; measure input and output power.	16–17 W: $\sim$60%; 1–2 W: $>$90%
Marker estimate is within $\pm$5 mm and $\pm3^\circ$.	Trigger `/start_detect` at known marker offsets; compare corrected ArUco estimates with fixture positions.	Offset error $\leq$0.5 mm
Final docking error is no more than 1 cm.	Run autonomous docking trials and measure coil offset.	TBD
STM32 motion-control commands are safe and repeatable.	Send low-speed `/cmd_vel` commands; measure wheel response, stopping error, and malformed-frame rejection.	Verified
Loss of valid docking command stops the vehicle.	Stop the docking node or publish marker-lost state; verify zero-velocity command and zero motor PWM.	TBD
ESP32 reliably forwards alignment data to DSP.	Send packets for 10 min and log checksum failures.	TBD
Final-result transfer latency is no more than 50 ms.	Timestamp Raspberry Pi result frame, TCP receipt, UART forwarding, and PWM update.	Verified
Output ripple is no more than 0.5 V peak-to-peak.	Measure receiver output with oscilloscope.	1.0 Vpp in CV; not met
Faults cause safe shutdown or hold behavior.	Inject fault conditions and verify PWM disable or update rejection.	TBD

Appendix B Coil Offset Calibration Data

Sheet2 of the coil-frequency workbook contains measured output power and optimal switching frequency for 44 coil-offset positions. These data were used to fit the DSP quadratic frequency model. Table 15 lists the measured calibration points.

Table 15: Coil offset calibration data. Offsets are in mm, output power is in W, and optimal frequency is in kHz.

Points 1–22					Points 23–44				
No.	x	y	P_o	f_o	No.	x	y	P_o	f_o
1	10.43	-1.93	16.70	62.20	23	-32.99	-51.98	18.00	63.37
2	26.99	-7.14	16.20	62.40	24	-41.34	-60.67	16.40	64.30
3	48.68	-2.11	17.10	63.00	25	-52.24	-72.59	16.40	65.22
4	73.29	-5.23	16.10	63.60	26	-60.65	-87.40	16.20	66.42
5	91.87	-27.96	16.00	64.70	27	-62.37	-82.87	16.20	66.20
6	93.40	-26.45	16.30	64.80	28	-46.98	-80.12	16.30	65.35
7	84.68	5.96	16.00	64.60	29	-30.55	-71.58	16.50	64.42
8	88.96	37.85	15.90	65.70	30	-20.02	-61.96	17.80	63.35
9	71.86	61.40	15.70	65.70	31	-14.99	-54.33	17.50	63.10
10	47.18	56.78	16.20	64.57	32	18.51	-41.93	16.90	62.42
11	-14.03	60.15	16.20	64.17	33	27.85	-42.48	17.00	62.50
12	-26.19	57.93	16.00	64.45	34	41.91	-39.53	17.09	62.72
13	-34.52	43.06	16.20	64.00	35	60.03	-41.92	18.80	63.45
14	-45.81	37.92	16.30	64.35	36	52.17	-60.42	17.20	63.55
15	-57.88	39.57	16.10	64.97	37	55.60	-71.71	16.40	63.97
16	-71.10	49.36	16.10	66.10	38	58.82	-55.84	18.00	63.30
17	-69.63	47.84	15.90	65.92	39	40.59	-38.40	17.00	62.60
18	-74.67	40.40	16.00	65.85	40	53.50	-27.97	17.10	62.77
19	-75.81	51.01	15.80	66.37	41	65.67	-20.04	17.30	62.95
20	-70.42	27.25	16.10	65.35	42	73.46	-22.92	18.40	63.35
21	-13.70	-31.46	16.90	62.50	43	63.18	-31.41	17.30	62.97
22	-23.59	-37.40	17.10	62.85	44	59.76	-24.59	17.20	62.77

References

[1] W. C. Brown, "The history of power transmission by radio waves," *IEEE Transactions on Microwave Theory and Techniques*, vol. 32, no. 9, pp. 1230–1242, 1984.

[2] X. Wu, H. Xu, J. Xiao, Y. Mo, N. Wu, and S. Chen, "Overview of wireless power supply technology for electric vehicles," in *2023 IEEE 6th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, 2023, pp. 66–69.

[3] M. Liu, X. Wang, and J. Xu, "Design methodology of SiC MOSFET based bidirectional CLLC resonant converter for wide battery voltage range," in *2021 IEEE Workshop on Wide Bandgap Power Devices and Applications in Asia (WiPDA Asia)*, 2021, pp. 423–427.

[4] X. Li and A. K. S. Bhat, "Analysis and design of high-frequency isolated dual-bridge series resonant DC/DC converter," *IEEE Transactions on Power Electronics*, vol. 25, no. 4, pp. 850–862, 2010.

[5] Raspberry Pi Foundation, *Raspberry pi 4 model B documentation*, 2024. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[6] OpenCV. "Detection of ArUco markers. "[Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[7] STMicroelectronics, *STM32 32-bit ARM Cortex MCUs reference manual*, 2024.

[8] IEEE. "IEEE Code of Ethics," Accessed: Feb. 8, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[9] International Commission on Non-Ionizing Radiation Protection, "Guidelines for limiting exposure to electromagnetic fields (100 kHz to 300 GHz)," *Health Physics*, vol. 118, no. 5, pp. 483–524, 2020.