ECE445 Senior Design Laboratory
Design Document
Design of a Mechatronic Physical Road-Crossing Game System
Team #47

Yuxuan Liu [yuxuan60]
Tianxi Zhu [tianxiz2]
Zihao Wu [zihao14]
Zhuo Li [zhuol4]
Advisor: Timothy Lee

April 2, 2026

Contents

1 Introduction

1.1 Problem Statement

Most games that students encounter today are implemented in purely digital environments. A mobile or computer game can create challenge and entertainment, but it does not show how sensing, mechanical motion, electrical power delivery, and embedded control must interact in a real physical system. This gap is especially important in mechatronics education, where students need to understand not only software logic, but also the constraints introduced by motors, sensors, friction, wiring, timing uncertainty, and repeated operation.

Our project addresses this educational gap by designing a mechatronic physical road-crossing game system. In the game, several traffic vehicles move continuously across a road-like platform while a player-controlled vehicle attempts to cross from a start area to a goal area without colliding. The project is intentionally simple in its gameplay, but technically rich in its implementation. The system must generate repeatable motion, detect collisions in real time, respond to user inputs with low latency, and recover cleanly after each round. These requirements make the project a good demonstration of integrated mechatronic design rather than a purely mechanical toy or a purely software exercise.

A practical design must also be robust enough for repeated classroom demonstrations. If the traffic vehicles drift, the collision logic is inconsistent, or the system response is delayed, then the educational value is reduced and the game becomes frustrating rather than instructive. The design therefore emphasizes repeatability, modularity, safety, and clear separation of subsystem responsibilities.

1.2 Solution Overview and Visual Aid

We propose a desk-scale physical game platform representing a multi-lane road. Each lane contains one traffic vehicle constrained to move along a fixed linear track. A player controls a separate vehicle beginning at the bottom of the platform and moves it forward in discrete steps using push buttons. The controller continuously supervises traffic motion, player motion, collision sensing, and game-state transitions. If the player reaches the goal region, the system declares success. If the player vehicle collides with any traffic vehicle, the system stops motion and declares failure. After a reset command, the system returns all moving elements to their starting states and begins a new round.

From a mechatronics perspective, the complete system integrates five functional layers. The first is the mechanical layer, consisting of the platform, track guides, mounts, and vehicle structures. The second is the actuation layer, which uses motors and motor-driver electronics to create repeatable motion. The third is the sensing layer, which includes collision sensors, a goal sensor, and local position or limit feedback. The fourth is the embedded control layer, implemented on a microcontroller that executes timing logic and finite-state behavior. The fifth is the user interface layer, including push buttons, LEDs, and optional buzzer indication. These layers must interact reliably or the overall system cannot perform as intended.

Figure 1 illustrates the intended use context of the road-crossing game. It highlights the relationship among the player, the moving traffic vehicles, and the system logic that determines success or failure.

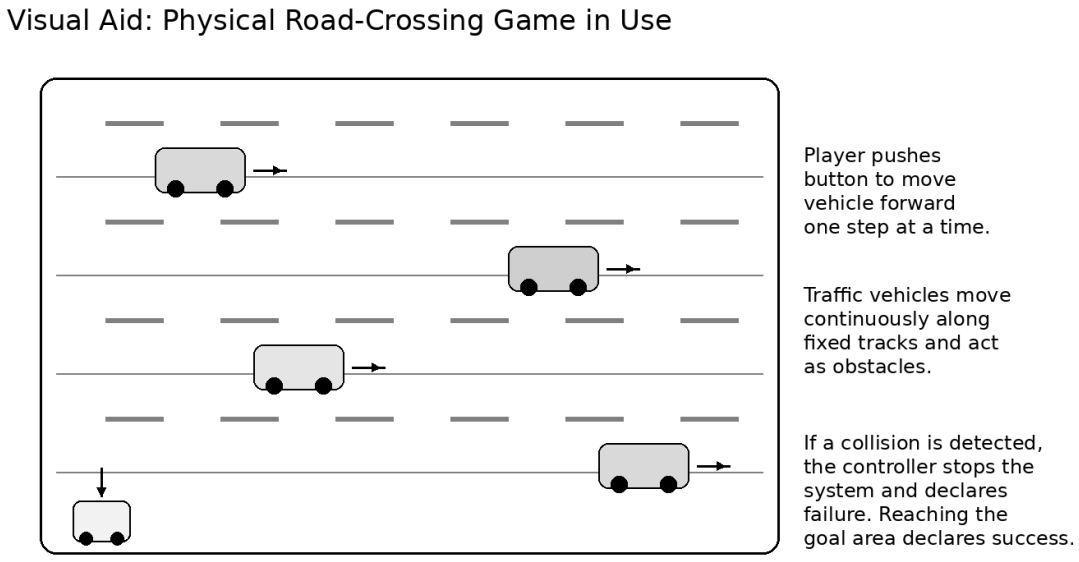

Figure 1: Visual aid of the mechatronic physical road-crossing game system in use.

1.3 High-Level Requirements List

The mechatronic physical road-crossing game system shall satisfy the following high-level requirements.

1. **Traffic motion performance:** The system shall drive at least four traffic vehicles along fixed tracks with adjustable speeds from 0.10 m/s to 0.50 m/s. For any selected nominal speed, the steady-state speed variation of each lane shall remain within ±5% over a 30 s observation interval.
2. **Gameplay response and detection:** The system shall respond to a valid player input within 100 ms, detect collision or goal events within 50 ms of occurrence, and correctly classify at least 95% of tested collision and non-collision scenarios.
3. **Reliability and demonstration readiness:** The platform shall operate for at least 30 minutes of repeated rounds without structural loosening, controller reset, loss of sensor functionality, or unacceptable drift of the traffic vehicles from their intended tracks.

2 Design

2.1 Block Diagram

The system is divided into five interacting modules: player input, controller and game-state logic, traffic actuation, sensing and event detection, and power and user interface. The controller is the coordination center. It receives push-button and sensor signals, determines the current system state, commands motor motion, and updates the user-visible outputs. Unlike a purely software

game, our design must explicitly manage the timing relationship between moving traffic vehicles and the stepwise motion of the player vehicle. That constraint makes the signal flow among the modules an essential part of the design, not just a software implementation detail.

Figure 2 shows the overall block diagram. The player-input subsystem sends commands to the controller. The controller drives both the traffic-motion and player-motion paths through motor drivers. The sensing subsystem sends collision, goal, and reset-related information back to the controller. The power subsystem provides separate rails for motors and logic, while the user-interface path displays status and fault information.

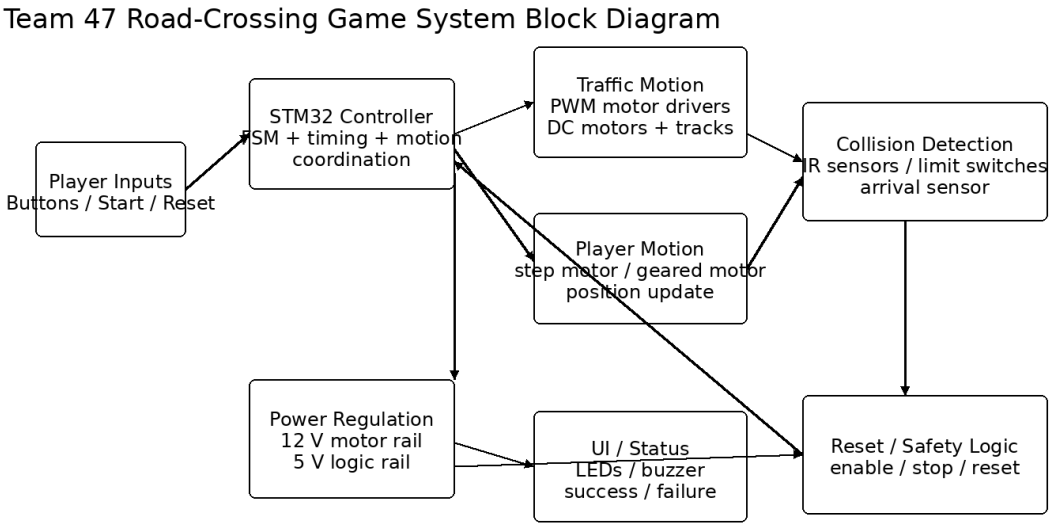

Figure 2: System block diagram for Team #47 road-crossing game system.

The block diagram also reflects a deliberate modular design choice. Traffic motion, player motion, and event detection are not combined into one monolithic circuit. Instead, each function is separated so that mechanical tuning, firmware testing, and verification can proceed independently. This approach reduces debugging complexity and allows the team to identify whether a failure originates from the sensor layer, the control logic, or the actuation path.

2.2 Player Vehicle Control Subsystem

The player vehicle control subsystem is responsible for converting human intent into repeatable forward motion of the player vehicle. Because the game must be intuitive for demonstration use, the player interaction must be simple and unambiguous. For this reason, we choose a discrete step-based control strategy rather than continuous joystick or analog speed control. Each valid button press advances the player vehicle by one fixed increment toward the goal region. A reset input returns the vehicle to the start state, and an optional start input enables the controller to begin a round only after all traffic lanes are active.

The decision to use discrete steps is motivated by both gameplay clarity and implementation simplicity. From the user perspective, discrete motion makes the game easy to understand: one press corresponds to one move. From the engineering perspective, it reduces the need for continuous closed-loop position control on the player axis. Instead of trying to hold an arbitrary analog position, the firmware only needs to move to a known increment and stop. This lowers software complexity, reduces the risk of overshoot, and makes verification more direct.

The player vehicle will be driven either by a small geared DC motor with a step-counting mechanism or by a stepper motor if higher repeatability is required after mechanical prototyping. The physical implementation uses a guided slot or rail so that the vehicle cannot yaw or drift laterally while moving forward. This is important because lateral motion error would affect the interpretation of collision zones and undermine the repeatability of the game logic.

A debounce function is included in firmware so that a single mechanical button press is not misinterpreted as multiple commands. The interrupt or polling interval will be selected such that a valid press is recognized quickly while bounce does not create false extra motion. The subsystem interacts with the controller through command and acknowledge signals. The controller issues a move command, the motion hardware executes the movement, and the position flag or internal move timer indicates completion.

Table 1: Player vehicle control subsystem requirements and verifications.

Requirement	**Verification Procedure and Success Criterion**
The player vehicle shall advance by 10 ± 2 mm for each valid movement command.	Issue 20 movement commands from the same start position and measure cumulative displacement with a ruler or caliper. The subsystem passes if the average step length is within the specified tolerance and no single step deviates by more than 3 mm.
The delay between a valid player button press and the onset of vehicle motion shall be less than 100 ms.	Observe the button signal and motor-enable signal on an oscilloscope or logic analyzer. The subsystem passes if the measured delay remains below 100 ms in at least 19 out of 20 trials.
A single physical button press shall not generate more than one motion command.	Perform 50 button presses of varying force and duration. The subsystem passes if no unintended multi-step response occurs during the test.

2.3 Traffic Vehicle Motion Subsystem

The traffic vehicle motion subsystem generates the dynamic obstacles that define the gameplay challenge. Each traffic vehicle moves laterally along a fixed track across one lane of the road. The tracks ensure constrained one-dimensional motion, which allows the controller to reason about lane occupation without needing complex planar localization. This fixed-track decision is one of the most important system-level simplifications in the project. It reduces mechanical uncertainty, keeps motor control manageable, and makes collision verification substantially easier.

Each lane is driven by its own motor channel. The baseline design uses DC motors and PWM-based speed control because the motion requirement is continuous rather than position-indexed. Each traffic lane may be assigned a different nominal speed to make the game more dynamic and to illustrate how the controller coordinates multiple moving elements in parallel. The speed range is

intentionally moderate. If the lanes move too slowly, the game loses challenge and the educational value of timing-based control is weakened. If the lanes move too fast, the player interface becomes frustrating and the sensing subsystem has less timing margin.

Mechanical repeatability is just as important as electrical control in this subsystem. The traffic vehicles must remain aligned with their guide rails, and the track structure must resist loosening during repeated operation. The design therefore includes lane guides, rigid motor mounts, and hard-stop or limit features to prevent vehicles from leaving the defined travel region. Depending on the final prototype, motion can be implemented as reciprocating travel or as continuous belt-driven circulation. For a first prototype, reciprocating motion between end points is easier to package and verify.

The traffic motion subsystem interacts with the controller in two ways. First, the controller sets or modulates lane speed through PWM duty cycle or driver commands. Second, the motion subsystem provides state feedback through end-stop sensors, timing-based lane position estimation, or initialization homing events. This feedback allows the controller to synchronize round start conditions and avoid accumulating motion drift over multiple rounds.

Table 2: Traffic vehicle motion subsystem requirements and verifications.

Requirement	**Verification Procedure and Success Criterion**
Each of at least four lanes shall support adjustable traffic speed from 0.10 m/s to 0.50 m/s.	Command three operating points per lane and measure travel time over a known distance. The subsystem passes if all measured speeds fall within the required range.
For a selected nominal operating speed, the steady-state speed variation shall remain within ±5% over 30 s.	Record lane travel time repeatedly over a 30 s interval. The subsystem passes if the computed speed remains within ±5% of nominal for all recorded segments.
Traffic vehicles shall remain mechanically constrained to their tracks for at least 30 minutes of repeated operation.	Run continuous motion for 30 minutes and inspect for derailment, mount loosening, or unacceptable rubbing. The subsystem passes if no lane leaves its guide or requires manual realignment.

2.4 Collision Detection and Game-State Subsystem

The collision detection and game-state subsystem determines whether the player has failed by colliding with traffic or succeeded by reaching the goal. This subsystem is central to the educational value of the project because it converts physical interaction into discrete logical outcomes. If event detection is unreliable, then the user cannot trust the system and the game ceases to function as a demonstration of embedded decision-making.

Our design uses a set of lane-level collision sensors combined with a dedicated goal sensor. Infrared break-beam or reflective IR sensors are appropriate because they offer fast response, low cost, and straightforward integration with a microcontroller. In locations where direct beam alignment is impractical, low-force contact switches can serve as backup or complementary detectors. The exact implementation may depend on mechanical packaging after the prototype layout is finalized, but the detection logic is the same: when the player vehicle occupies a lane region at the same time as the traffic vehicle in that lane, the controller must recognize a collision condition and stop the

round.

The game-state portion of this subsystem is implemented in firmware as a finite-state machine. The principal states are *Idle*, *Running*, *Failure*, and *Success*. In the Idle state, the system waits for initialization or user start. In the Running state, traffic moves and player commands are accepted. In the Failure state, a collision has been detected; motors are stopped and the failure indication is displayed. In the Success state, the player has reached the goal region; traffic is halted or ignored and the success indication is displayed. Reset returns the system to Idle or directly into a new Running state depending on the final user-interface choice.

This subsystem is intentionally separated from the raw sensing hardware because the decision thresholds and state transitions are design work performed by the team, not merely an off-the-shelf sensor function. By separating the logical event layer from the sensor hardware layer, we can verify not only whether a sensor changes state, but also whether the controller interprets those changes correctly under realistic timing conditions.

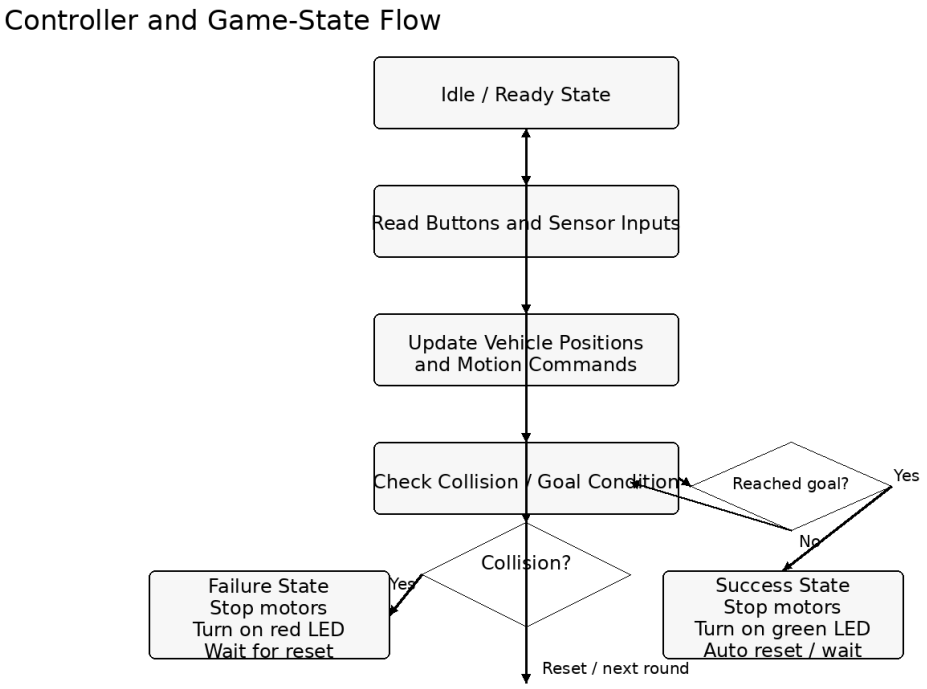

Figure 3: Controller and game-state flow used to classify success and failure events.

Table 3: Collision detection and game-state subsystem requirements and verifications.

Requirement	Verification Procedure and Success Criterion
The system shall detect collision and goal events within 50 ms of physical occurrence.	Trigger collision and goal events while observing the sensor output and controller state-indication output. The subsystem passes if the measured delay is less than 50 ms in all tested cases.
The controller shall correctly classify at least 95% of 100 tested collision and non-collision scenarios.	Run 100 scripted gameplay cases including near misses, true collisions, and successful crossings. The subsystem passes if at least 95 cases are classified correctly.
After entering success or failure state, the controller shall stop accepting movement commands until reset.	Issue repeated player commands after a detected event. The subsystem passes if no additional motion command is executed until reset logic is asserted.

2.5 Controller, Power, and User Interface Subsystem

The controller, power, and user interface subsystem forms the electronic backbone of the project. A microcontroller such as an STM32 is responsible for coordinating the timing of every round, issuing motor commands, reading sensors, and driving the visible status indicators. Unlike the player or traffic subsystems, which are strongly tied to a single physical mechanism, this subsystem supervises the entire platform and must therefore be designed for clarity and robustness.

Firmware is organized around a finite-state machine and a periodic main loop. The main loop samples debounced button inputs, checks sensor flags, updates timers, and refreshes user outputs. Interrupts may be used for time-critical events such as edge-triggered buttons or end-stop sensing, but the overall logic remains understandable because high-level decisions are still handled by explicit state transitions. This architecture is chosen to keep the code deterministic and maintainable for a student prototype.

The power portion of the subsystem includes separate rails for logic and motors. A regulated 5 V rail powers the microcontroller and low-power sensors, while a higher-voltage rail such as 12 V powers the motors through their driver stages. Separating these paths reduces the likelihood that motor startup current or switching noise will brown out the control board. It also makes debugging easier because controller resets can be isolated from motor disturbances during bench testing.

The user-interface portion is intentionally simple. At minimum, the interface includes movement input buttons, a reset button, and status LEDs indicating Idle, Running, Success, or Failure. A buzzer may also be added to provide immediate audible feedback during demonstrations. The goal is not to create a sophisticated interface, but to make the operating state unambiguous to both the player and an observer.

Table 4: Controller, power, and user-interface subsystem requirements and verifications.

Requirement	Verification Procedure and Success Criterion
The main control loop shall update all critical logic within 50 ms.	Instrument firmware with timing markers and measure loop period over 5 minutes. The subsystem passes if the worst-case critical loop interval is below 50 ms.
The controller shall operate continuously for 30 minutes without unintended reset or state corruption.	Run repeated rounds for 30 minutes and log controller status. The subsystem passes if no spontaneous reset or invalid state transition occurs.
The status interface shall clearly indicate Idle, Running, Success, and Failure states.	Exercise each state and verify unique LED or buzzer indication for each. The subsystem passes if an observer can distinguish all four states without referencing firmware logs.

2.6 Physical Implementation and Integration

The physical implementation is designed as a layered prototype rather than a tightly packed consumer product. The top layer contains the visible road platform, lane guides, traffic vehicles, and player path. Beneath the platform, motor mounts, sensor brackets, and wiring harnesses are attached to a rigid support frame. Electronics are grouped near one side of the enclosure or under-platform service zone so that debugging access is available during integration. This arrangement reflects the priorities of a senior design prototype: observability and serviceability are more important than maximum packaging density.

One important integration decision is to keep the high-current motor wiring separated from the low-level sensor and logic wiring wherever possible. This reduces noise coupling and makes troubleshooting more manageable. Connectors for sensors and motors will be keyed or labeled to minimize assembly mistakes. If the final build includes removable modules for each lane, then one lane can be serviced or retuned without disassembling the entire platform.

A second important integration choice is to design for repeatable reset and restart. In a classroom or demo setting, the system must recover quickly between rounds. The player vehicle should be able to return to its start region without requiring manual repositioning. Likewise, traffic lanes should begin from known operating conditions after reset, whether that is achieved by homing sensors, timed initialization, or mechanical end references.

Figure 4 illustrates the physical implementation concept. It shows the relationship between the top platform, under-platform electronics, motion lanes, and service-access regions.

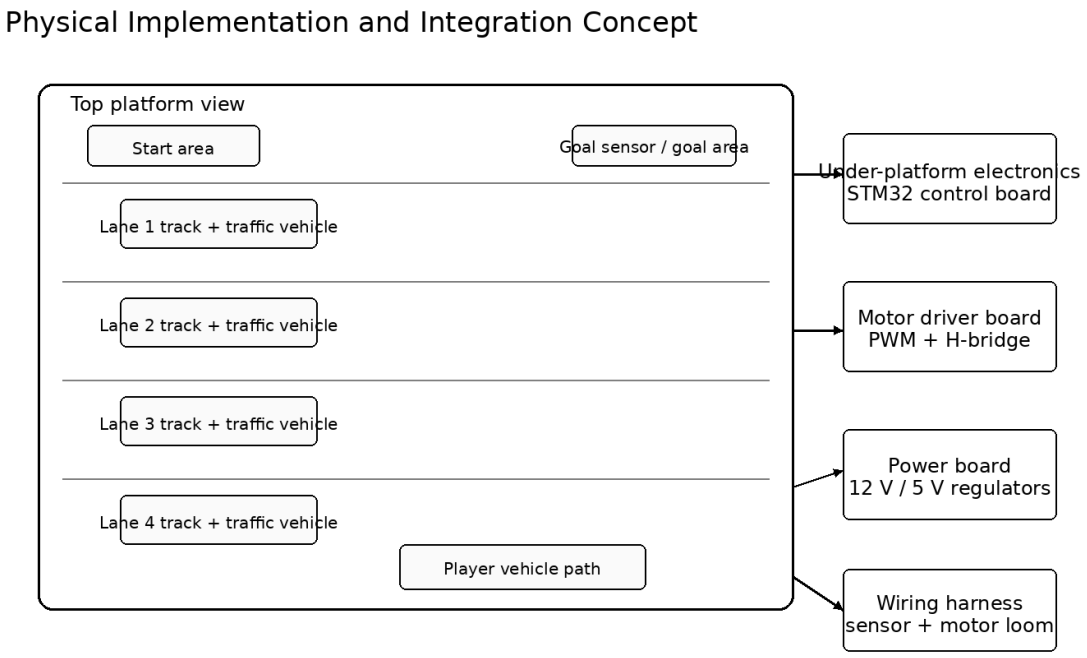

Figure 4: Physical implementation and integration concept for the desk-scale prototype.

2.7 Tolerance Analysis

The most critical dynamic quantity in the game is the time margin available for the player vehicle to enter or clear a lane before a traffic vehicle arrives. If this timing margin becomes too sensitive to speed error, step-length variation, or sensor delay, then the gameplay becomes inconsistent and the system no longer demonstrates repeatable mechatronic behavior. We therefore treat the crossing-time margin as the key tolerance-analysis target.

Let g be the effective safe gap between the player and a traffic vehicle when the lane is considered clear, and let v_t be the traffic vehicle speed. Then the available time window before the gap closes is approximately

$$t_{\text{gap}} = \frac{g}{v_t}.$$

For our prototype-scale analysis, we use a nominal effective safe gap of $g = 60$ mm and a traffic speed range of 0.10 to 0.50 m/s. At the maximum traffic speed, the nominal available window is

$$t_{\text{gap}} = \frac{0.060}{0.50} = 0.12 \text{ s},$$

while at the minimum traffic speed it is

$$t_{\text{gap}} = \frac{0.060}{0.10} = 0.60 \text{ s}.$$

These values show that the difficulty of the game changes significantly with lane speed, which is desirable from a gameplay perspective but only if the system remains predictable. To capture component and implementation variation, we also consider a gap uncertainty of ±5 mm caused by sensor threshold placement and vehicle geometry, together with a speed uncertainty of ±5% due to motor variation. Under those assumptions, the worst-case timing margin for the fastest lane becomes

$$t_{\text{worst}} = \frac{0.055}{0.50 \times 1.05} \approx 0.105 \text{ s}.$$

This is still larger than the 50 ms event-detection requirement and roughly comparable to the 100 ms player-command response requirement. That comparison is important: it indicates that the control and sensing design must stay within specification or the fastest lane becomes unfairly difficult. Conversely, the lower-speed lanes retain much larger timing margin and therefore serve as easier gameplay conditions.

Figure 5 plots the nominal, best-case, and worst-case crossing-time window over the traffic-speed range. The figure shows that the game remains feasible over the intended operating range, but it also justifies why lane speed should be capped and why controller and sensor latency must remain tightly controlled.

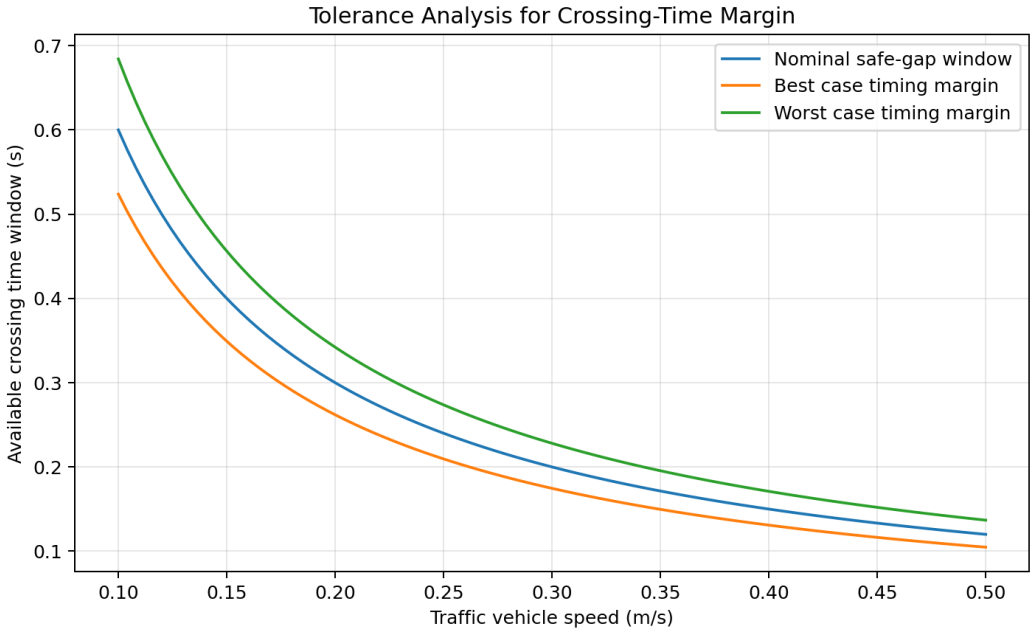

Figure 5: Predicted crossing-time margin under nominal and tolerance-bounded traffic conditions.

3 Cost

Table 5 lists the estimated material cost for one prototype. Prices are approximate and reflect single-quantity or low-volume student sourcing rather than production purchasing.

Table 5: Estimated bill of materials for Team #47 prototype.

Part	Qty.	Unit Cost (USD)	Notes
STM32 development board or equivalent controller	1	18	Central control and firmware development
DC motors for traffic lanes	4	9	One per lane
Motor driver modules	4	6	PWM-based motor control
Player vehicle motor / stepper assembly	1	16	Discrete player motion path
Infrared sensors / break-beam sensors	6	3	Lane collision and goal detection
Buttons, LEDs, buzzer, interface parts	1 set	10	User interaction and status display
12 V power adapter and 5 V regulator hardware	1 set	18	Logic and motor rails
Acrylic, plywood, or 3D-printed platform parts	1 set	35	Mechanical frame and guides
Belts, pulleys, rails, or linkages	1 set	24	Motion transmission hardware
Wiring, connectors, headers, prototyping material	1 set	20	Integration and assembly
Fasteners, brackets, and mechanical spares	1 set	15	Structural support and rework margin
Estimated total		**192**	

4 Schedule

The remaining project schedule is organized by week and by team member so that each week has concrete ownership. This mirrors the organization style used in the sample design document and ensures that implementation, integration, and testing proceed in parallel rather than sequentially.

Table 6: Team #47 implementation schedule.

Week	Yuxuan Liu	Tianxi Zhu	Wuzihao	Zhuo Li
04/06	Finalize controller architecture and state definitions.	Finalize traffic-lane mechanical concept and motor selection.	Finalize collision sensor placement and lane geometry.	Finalize player-path mechanism and reset strategy.
04/13	Implement firmware framework for state machine, timers, and input handling.	Assemble first traffic lane prototype and characterize speed range.	Build and bench-test collision and goal sensor circuits.	Assemble player vehicle drive mechanism and verify step repeatability.

04/20	Integrate motor-control commands and basic round logic.	Replicate remaining traffic lanes and refine guide alignment.	Integrate sensors with controller and validate event timing.	Integrate player motion with controller and tune button debounce.
04/27	Implement reset, success, and failure behaviors with visible status outputs.	Run 30-minute lane-motion reliability test and correct mechanical issues.	Perform collision-detection accuracy tests and improve thresholding.	Complete platform assembly and under-platform cable routing.
05/04	Run full end-to-end gameplay tests and tune controller timing.	Finalize traffic speed calibration for all lanes.	Document verification results and create requirement tables.	Finalize physical packaging, labels, and demonstration readiness.
05/11	Prepare final demo script and presentation material.	Prepare hardware spare parts and safety checklist.	Prepare final report figures and test evidence.	Support final integration and document formatting.

5 Ethics and Safety

5.1 Ethics

The project is intended as an educational mechatronic demonstration, so its ethical obligation is to present system behavior honestly and to prioritize user safety over game difficulty or visual appeal. We will follow the spirit of the IEEE Code of Ethics in both design and reporting. In practice, this means that we will report measured performance rather than idealized expectations, acknowledge the limits of the prototype, and avoid claiming reliability or safety margins that we have not verified experimentally.

We also consider fairness in the design of the game itself. Because the system is used for demonstration and learning, the player should be able to trust that success or failure is determined by real physical interaction rather than arbitrary hidden logic. This is one reason we emphasize repeatable lane motion and measurable sensor thresholds. A physically unfair game would undermine the educational purpose of the prototype.

Finally, we consider resource responsibility. Although this is not a high-power system, the design still uses motors, electronic boards, plastics, and prototyping materials. Where possible, the structure is designed to be reusable across testing cycles rather than discarded after a single demonstration. Components such as the controller board, drivers, and sensors can be recovered and reused in future mechatronics projects.

5.2 Safety

The primary hazards in this project arise from moving mechanisms, exposed wiring, and unintended motion during testing. To reduce these risks, the system will operate at low voltage, with logic circuitry on a regulated low-voltage rail and motion hardware on a modest motor supply. High-current or moving components will remain under the platform or inside protected regions so that users do not need to touch them during normal gameplay.

An emergency stop or immediate disable function will be included in firmware and hardware control logic so that all motion can be halted during debugging or demonstration. Before each demonstration, the team will inspect lane guides, motor mounts, wiring strain relief, and exposed connectors. This matters because a mechanically loose traffic vehicle can create both inaccurate gameplay and a physical pinch hazard.

The project is a student prototype rather than a certified commercial product, but the design still follows common good practice for low-voltage embedded systems: separate power domains for logic and motors, protected wiring paths, controlled reset behavior, and clear visible indication of current operating state. These measures support safe classroom use and reduce the chance of damage during repeated testing.

6 References

1. R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 13th ed. Pearson, 2016.
2. W. Bolton, *Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering*, 7th ed. Pearson, 2019.
3. IEEE, "IEEE Code of Ethics," IEEE. [Online]. Available: `https://www.ieee.org/about/corporate/governance/p7-8.html`. [Accessed: Apr. 2, 2026].
4. J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 4th ed. Pearson, 2017.