

**LG Software Architectures Training Program - Project Description**

**Automatic License Plate Recognition System**

Project Overview

Law enforcement today frequently utilize automated license plate recognition (ALPR) technologies to enhance their enforcement abilities, expand data collection, and to provide the capability for law enforcement to automatically compare vehicle license plates against lists of stolen, wanted, and other vehicles of interest while on patrol. The goal of this project is to design a robust and secure ALPR system for use by law enforcement for the project sponsor, WeCatchEm Inc., that will launch this new product later this year.  Key motivators for this project are:

* Select the best architecture/solution based on the results of this competition among the teams in this course.
* Ensure secure system communication with the backend server that will be accessed from the police vehicle via a 5G cellular network.
* Fast and accurate license plate lookup and response times.
* User interface that is intuitive that requires minimal human interaction.

ALPR Overview

As vehicles pass through the ALPR camera field of view a picture is taken and a series of algorithms are performed on the image. See Figure 1. First the image is evaluated to determine if a plate is visible, and if so, the plate is then isolated from the image and the alphanumeric characters are converted into a computer readable format.  Generally, there are set of basic algorithms that the software performs to identify and read a license plate and the accuracy of the system is typically driven by the sophistication and complexity of each of these algorithms.

1. Detection - Finds potential license plate regions for further processing
2. Character Analysis - Finds character-sized "blobs" in the plate region
3. Plate Edges - Finds the edges/shape of the license plate
4. Deskew - Transforms the perspective to a straight-on camera view based on the typical license plate sizes
5. Character Segmentation – isolates characters so that they can be processed individually
6. Optical Character Recognition - Analyzes each character image and provides possible letters/confidences
7. Syntactical/Geometrical analysis – Check characters and positions against state-specific formats

In this project, each team will create a software architecture and implement the software for a ALPR system that utilizes a laptop that will be located in the police vehicle and communicate via a network to a remote backend license plate database contains records with  the vehicle owner, make, model, color and status.

All teams will interact and communicate their client related questions or concerns with a single WeCatchEm point of contact.



*Figure 1: License Plate Recognition*

System Description

The goal of this project is to design and implement the software for a ALPR system that provides the following functionality.

*Laptop Application*

The laptop application shall provide the following functionality:

1. Login and authenticate user locally and to the backend license plate database lookup system.
2. Provide secure communication between the laptop application and to the backend license plate database lookup system.
3. Read images from a playback file and identify license plates for evaluation.
4. Perform the ALPR function in real-time while maintaining a frame rate of at least 25fps.
5. Query the backend license plate server for details about the vehicle. The user must be alerted for vehicles that are stolen, the owner is wanted (criminal), or if it is a vehicle of interest (expired registration, unpaid tickets, owner is missing).  Alerts must contain reason, owner name, address, and vehicle make, model and color along with the isolated plate image and the recognized license plate number for operator comparison.
6. If a license plate does not generate an alert, then the UI must display the last recognized plate image, the recognized license plate number and vehicle make, model and color so the operator can visually check if the plate matches the vehicle if desired.
7. An area in the UI that contains the playback view at all times.
8. The UI shall display the computed camera / playback frames per second, average time per frame, jitter and frame number.
9. The ability to detect network connectivity issues with the backend server within 5 seconds and automatically resolve the communication issue if possible.
10. Alert operator of any communication errors or failures.

*Server Application*

The server application shall provide the following functionality

1. Provide license plate queries
2. Ensure secure communication with the laptop applications
3. Authenticate remote laptop users
4. Support multiple users
5. Return the best match license plate if there is not an exact match that includes a configurable minimum confidence threshold to support a partial match.
6. Track the average number of queries per second for each user and overall queries per second, for all users.
7. Track the number partial matches and no matches for each user and all users
8. Support configurable values via a configuration file.

The vehicle registration database on the server contains the following information for each vehicle:

|  |
| --- |
| License Plate Number |
| Status - Owner Wanted / Stolen / Unpaid Fines – Tow / No Wants/Warrants |
| Registration Expiration |
| Owner Name |
| Owner Date of Birth |
| Owner Street Address / Location |
| Owner City, State and Zip Code |
| Vehicle Year of Manufacture |
| Vehicle Make |
| Vehicle Model |
| Vehicle Color |

*Table 1: Vehicle Registration Database*

System Hardware

*Team Laptops*

Teams will use their laptop to develop and execute both the user application and the backend license plate database server.  Each application will be run on separate laptops and communicate via Wi-Fi.

Required Software

Software will be developed using Visual Studio 2022 Community Edition running on Windows 10/11 with OpenCV 4.5.5.

Sample Code Archive

An example ALPR C++ application that uses OpenALPR (with all source code) is provided as shown in Figure 1. Please note this example application should NOT be considered an exemplar architecture or code.  It merely provides a means for demonstrating how some basic features could be implemented and should not be used as an architecture to build upon or extend. Some example code may also not be implemented in the best or most efficient way, may not be complete.

**Project Deliverables**

There are three milestones required for this project (each of which will be graded independently):

1. **Requirements, plan and risks**: For the first milestone the team will turn in the prioritized architectural drivers, any draft design decisions they are considering, the technical risks identified, and the experimentation plans (or any results already achieved).
2. **Experimentation results, design, plan for construction**: The team will turn in the results of experiments conducted, a design description with different architecture views, and the plan for construction.
3. **Demo and lessons learned**: For the final milestone, the team will demo the completed system, and present the lessons learned.

Milestone 1

The submission can be informal documents describing the team’s current understanding of the items listed below.  Team mentors will be meeting with the teams to ask follow up questions.

* Project Plan
  + We will see if the design is sound enough to guide construction.
  + We will evaluate when the team will be able to determine how well the design supports the architectural drivers (based on the planned activities).
  + The plan should describe the overall architecture, the division of roles, the specific tasks planned, and the associated milestones.
* Architectural Drivers
  + Are the QA requirements “actionable”?  In other words, are they expressed in such a way that the team will be able to determine if a given design supports these drivers or not?
  + Do the drivers seem to relate to the overall objectives of the project?
  + Are the measures clearly derived from the overall goals of the project?
  + Are the functional requirements understood?
  + Is there a mechanism for prioritizing the requirements?
* Risk Assessment/Planned Experiments
  + What are the technical and non-technical risks? How do you assess each risk with respect to probability and impact in a H-M-L scale?
  + Are the open questions/issues clearly related to things that will affect the outcome of the project?
  + Have there been any actions identified to address the open questions/issues?
  + Are the experiments concretely articulated?
  + Is it clear what question/issue is being addressed by the experiments?
  + Will it be clear when the experiments are complete?
* Architectural Approaches
  + What is the overview-level description of the architecture?
  + What are the main architectural approaches (tactics, patterns, design strategies) in your solution?
  + Are the architectural approaches clearly related to the drivers (will they likely impact the properties of interest)?

Milestone 2

Again, the submission can be informal documents describing the team’s current understanding of the items listed below.  The mentors will be meeting with the teams to ask follow up questions.

* Project Plan
  + How has the plan changed?
  + Has the team been actively assessing risk and updating the plan accordingly?
  + Does the team have a plan for any remaining significant issues/risks
  + Does the team have a reasonable construction plan?
* Experiments/Results
  + What experiments have been conducted?
  + Have the results of the experiments addressed the open questions/issues?
  + What experiments remain?
  + Are the experiments focused on issues relevant to the overall goals of the system?
* Architecture
  + What is the architecture in terms of the organization of code units and their dependencies? (The team shall create a module view of the architecture.)
  + What is the architecture in terms of components and connectors (runtime perspective)? (The team shall create a C&C/runtime view of the architecture.)
  + What is the architecture in terms of the supporting infrastructure (deployment perspective)? (The team shall create a deployment view highlighting component allocation to hardware elements and communication channels.)
  + Have the experiments led to a refinement of the architecture?
  + Does the team understand the architectural approaches they have/will realize?
  + Do the architectural approaches align with the goals of the system?
  + Are there significant concerns that have not been addressed?
  + Has the architecture been evaluated?

Milestone 3 - Final Demo and Presentation

For the final deliverable there will be both a team presentation and a demonstration of the final system.

*Presentation (150 points)*

The presentation should cover:

* Quality attribute requirements for the system.
* Architecture description showing architecture views and highlighting key architecturapproaches adopted and their rationale.
* Description and results of the architecture evaluation activities.
* Lessons learned (what went right, what went wrong, what would you have done differently).

*Demo (350 points)*

There will be a live demo of the system using the provided video of license plates.  The criteria for evaluating the demo will be:

* Accuracy of detection
* Detection time
* Playback speed

**Note: Any application crash or improper behavior during the demonstration will result in a 30-point deduction for each occurrence.**

Grading

The grading will be as follows:

* Milestone 1 —> 25% of the project grade
* Milestone 2 —> 25% of the project grade
* Milestone 3 —> 50% of the project grade

 Your team should produce a comprehensive architecture design, presentation, and system demonstration:

Design Document

Your team will produce a design document that addresses the following:

* Description of the project context. It should include a context diagram.
* Brief description of the architectural drivers. For quality attributes, focus on the high priority quality attributes. Describe them using clear quality attribute scenarios as presented in class. Prioritize your quality attributes. The architectural drivers should be used as a basis to design the architecture, reason about alternatives, and evaluate the fitness of the architecture.
* At least one module view showing the organization of code units and their dependencies. Be sure to use a template for architecture views. Complement structural diagrams with behavior diagrams as needed.
* At least one runtime/C&C view showing components and connectors and using the architecture view template.   Be sure to include sufficient detail in order to analyze the extent to which the key drivers are supported or inhibited. Complement structural diagrams with behavior diagrams as needed.
* A deployment view of the architecture to describe the hardware elements, how are they connected, and what coarse-grained components are allocated to each hardware element. Be sure to use the same view template.
* We expect you to create a few ADRs, one for each key design decision. In your architecture views, you can use the Rationale section to point to the ADRs. The ADRs should argue why your design is fit for purpose with respect to the architectural drivers.

You are free to structure your design document anyway you like, but you should follow the documentation principles presented in class and in the readings.