Phase II plan:

Preprocessing, postprocessing and optimization:

Video processing, which is the trickiest part of our project, is bound to be computationally expensive. To reduce the amount of processing to be done, I propose an idea:

i) Create 2 databases:
   a. A database of vehicle information (the database that would be present on the vehicle registrar's office's system)
   b. A database of vehicle images extracted and classified by YOLO indexed by their class labels (i.e. type of vehicle (car, bus, etc.). We could also add indices to other features if they are cheap to identify (color for example)
      i. This database shall also hold a table that maps the vehicles to their license plates once a query has been received.

ii) The second database will be dynamically updated in real time, so we need to study the feasibility of that. (Although it is my initial expectation that such a system is feasible (youtube and facebook's live video is an example).

iii) We need to design a method to remove redundant vehicle images (as a video feed has multiple frames of the same vehicle). A fast clustering algorithm will be required where we cluster images from same video feed and just keep one image from one cluster.

iv) A query system that allows users to or automatically generates a feature vector **x** which includes the license plate number as an attribute. (One can imagine this being automatically generated from the police department's F.I.R.s for stolen vehicles).

v) If the license plate has already been extracted in previous runs of the algorithm, its information is returned immediately. If not, it proceeds forward.

vi) The second database shall then be filtered first by the vehicle type which will significantly reduce the search space.

vii) If in step (ia) other features have been successfully extracted cheaply, then the database shall be further filtered based on those features.

viii) The images thus filtered will be fed into our original algorithm that extracts the license plate number.

ix) Again, redundant vehicles shall be removed.

x) The vehicle that match the feature set **x** shall be returned and also stored in the table of the second database as indicated in (i.b.i).

Project task division (Subject to change):

Dactar:

Write a clear interface specification for the vehicle extraction subsystem (YOLO).

Design and implement the YOLO subsystem. (We can work on foreign vehicles, whatever is easier.)

Try to extract as many features as possible.

Jeevan:

Work on the license plate extraction subsystem. This is the toughest part of our project, so everyone will collaborate, but you shall take lead here.

Write the interface specification for the subsystem.

Raju:

Be a flask maestro. Specifically you should study database access and video streaming with flask/python.

Take lead on the frontend. Ask us questions, irritate us with it if need be. Imagine you are the engineer and we three are your clients who want a website made.

Baagh:

Design a framework for all the pieces of the system to fit in as plugins.

Research and work on the optimization tasks mentioned above.

Some ideas:

i) Regarding License Plate extraction subsystem:
- If the vehicle extraction subsystem or the provided query also provides color information, then we could just mask the image with a filter so that the color is masked to black. Now, we could use some intensity metric to extract clusters of pixels that are left. If we can assume all license plates have the same color (black on white for example), we could further eliminate other pixel regions based on the color information.
- We could also do scaling and straightening of the extracted vehicle image so that the license plate approximately falls in the same region for every image. If this is possible, we can just perform our computation on the region only.
- We could extract images of vehicles classified by YOLO only when the bounding box of the vehicle just crosses an artificial horizontal line on the video. This will ensure that images will be of approximately the same scale. Then we might find some heuristic distance based on the type of vehicle to find an approximate region of the license plate. For example for a car image, the license plate might be approximately 50 pixels up from the lower boundary.
- Or, we could just scale the images based on the size of the bounding box. (this might be the simplest approach).

A framework for working on the project:

1) We need a chart that has the following three columns: (we can paste this at LeapFrog somewhere)

To Be Done	In progress	Completed

Sticky notes shall be pasted on this chart.

For each subsystem:

i) Define the input and output interfaces and commit that these interfaces shall not change later. This will ensure that others working on another system with a dependency will not be affected by any internal change in the subsystem.
ii) Prototype
iii) Functional test (to see if it works or not)
iv) Non-functional test (benchmark of performance, either from the algorithm's theory or a practical benchmark).
v) Integration into the input-output interface.
vi) Shift the sticky note to completed and push the code to the online repository. (The online repository discussed later).

Work plan:

Project meeting:

Case I: if we have classes/labs we must attend that day.

Every day 30 minutes before the first period/lab we all agree to attend if we do.

Case II: On days with insignificant classes

i. If the work to be done doesn't require collaboration (eg. implementation of a particular algorithm)
   Stay at home and do it.
ii. If the work absolutely requires collaboration (eg. designing the frontend)

   Meet up either at leapfrog or somewhere outside (NOT on campus. All kinds of shit going on, so need to focus)

Configuration management plan:

The github repository for the project will have folders on the root directory for each phase of the block diagram. There shall be one and only one administrator of one particular folder at a time and only he can edit any content inside. Others have only read/use access.