Udacity System Integration Project

Team JohnyCab

Team Member	Email Address
Muhammad Asani	asani_m@yahoo.ca
Bruno Guisard	bruno.guisard@gmail.com
Ian Hailey	ian@hailey.dk
Robin Reckman	robin.reckmann@web.de
Aaron Yang	aaron.yang@gmail.com

Solution Overview

Waypoint Updater

Configurable Parameters

Parameter Name	Description
/waypoint_loader/velocity	Speed limit in kph
/waypoint_loader/approach_velocity	Speed limit in kph as the vehicle aproaches a stop line
/tl_detector/max_detect_distance	Maximum detection distance for the lights in m
/waypoint_updater/update_rate	Main loop update rate Hz

Subscribed Topics

Topic Name	Description
/current_pose	Recieves the current position of the car
/base_waypoints	Recieves the waypoint list
/traffic_waypoint	Recieves the waypoint index for the next possible Stop Line corresponding to the next Traffic Light
/traffic_state	Recieves the status of the next traffic light
/obstacle_waypoint	Recieves the waypoint for the next obstacle

Published Topics

Topic Name	Description
/final_waypoints	Publishes the forward loking waypoints including target velocities
/current_waypoint_index	Publishes the waypoint nearest to the vehicle

Closest Waypoint Detection

In order to generate the correct points in front of the vehicle used by the waypoint follower, we have to compute the closest waypoint to the vehicle. Since this calculation is done in every time step, the algorithm should run as efficiently as possible. Because of this reason, we decided to use a kd-tree (k-dimensional tree) as a space-partitioning data structure. In a setup-step all track waypoints are recursively split into two subsets, creating a tree structure. Searching for a closest point in this tree can then be done much faster and efficiently compared to a simple brute force approach. The closest waypoint to the vehicle may be located behind the vehicle. This point should be discarded and the next waypoint in driving direction should be taken instead. To detect this condition, the global angle between the closest point and the vehicle is compared to the heading of the vehicle. If the difference is larger than 90 degrees, the point is located behind the vehicle.

Main Loop

On each itteration of the waypoint updater the forward looking waypoints are selected based on the current pose of the vehicle. If we are approching a stop line and need to stop each chosen waypoint's target velocity is adjusted downward relative to the target stop distance.

Drive By Wire (DBW)

Configurable Parameters

Parameter Name	Description
~throttle_scale	Adjusts the gain of the throttle commands to accomodate variation between the simulation and Carla

Subscribed Topics

Topic Name	Description
/vehicle/dbw_enabled	Recieves the current position of the car
/twist_cmd	Recieves the current twist commands from the waypoint follower
/current_velocity	Recieves the vehicles current velocity

Published Topics

Topic Name	Description
/vehicle/steering_cmd	Publishes the target steering command
/vehicle/throttle_cmd	Publishes the target throttle command
/vehicle/brake_cmd	Publishes the target brake command

Brake & Throttle

One PID controller for throttle is instantiated with the parameters: kp, ki, kd, deceleration limit, and acceleration limit. The kp, ki, and kd values were determined by trial and error.
During each loop if the DBW module is enabled, the following steps are done:

1. Track the delta time from the previous cycle
2. Get the difference between the current velocity and the target velocity.
3. Get the value from the PID controller using the parameters obtained in the previous two steps.
4. If we need to speed up - then the value from the PID controller is used directly as the throttle value and the braking value will be zero.
5. If we need to slow down - then the value from the PID controller is used for the braking, after converting to torque by multiplying the vehicle mass and wheel radius. This is for all delta velocities below the brake deadband value. The throttle value will be zero in this case.
6. If the target velocity and change in velocity are close to 0 (less than 0.1), then we completely apply the brake. The throttle value will be zero in this case.

There is an additional parameter, throttle_scale, which will scale the throttle value for Carla. For the simulation, throttle_scale will be 1.0. For Carla, throttle_scale will be 0.025. This was the value inferred from running dbw_test.

Steering

One YawController is instantiated with the parameters: wheel base, steering ratio, minimum velocity, maximum lateral acceleration, and maximum steering angle.

On every update cycle that the DBW module is enabled, the next steering value is obtained from the yaw controller based on the current yaw, current velocity, and target velocity.

Traffic Light Detector

Configurable Parameters

Parameter Name	Description
/waypoint_loader/velocity	Speed limit in kph
/waypoint_loader/approach_velocity	Speed limit in kph as the vehicle aproaches a stop line
/tl_detector/max_detect_distance	Maximum detection distance for the lights in m
/waypoint_updater/update_rate	Main loop update rate Hz

Subscribed Topics

Topic Name	Description
/current_pose	Recieves the current position of the car
/base_waypoints	Recieves the waypoint list
/current_waypoint_index	Recieves the waypoint nearest to the vehicle
/vehicle/traffic_lights	Recieves the traffic light locations
/image_color	Recieves the front facing vehicle camera data

Published Topics

Topic Name	Description
/traffic_waypoint	Publishes the waypoint index for the next possible Stop Line corresponding to the next Traffic Light
/traffic_state	Publishes the status of the next traffic light

Initialisation

Before the main loop starts the traffic light and stop line positions are converted into waypoint arrays. Another array is created mapping stop lines to associated their lights (if one exists).

Image Processing

On reception of a new front facing image the current distance to the next stop line and/or light is calculated and if less than the maximum detection range we publish this to traffic_waypoint. Assuming we are approaching a light we attempt to classify its state and publish this to traffic_state.

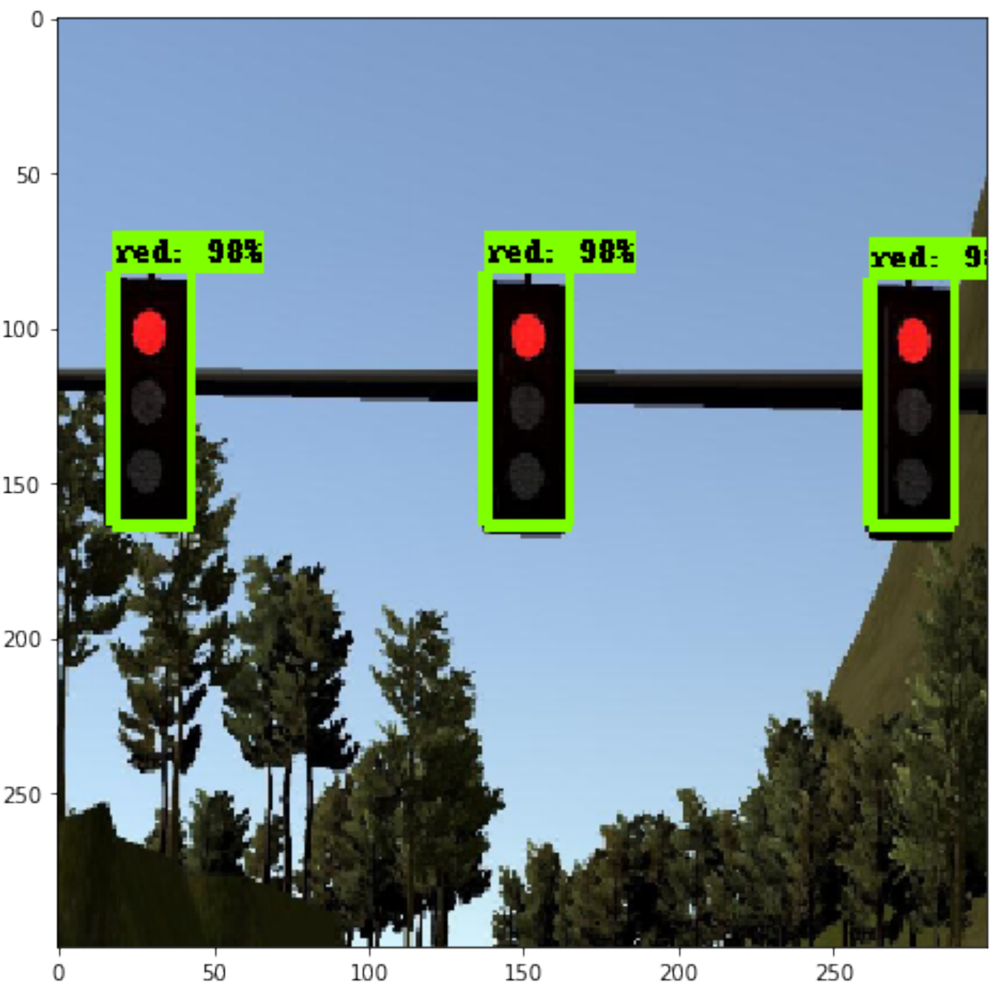
Light Classification

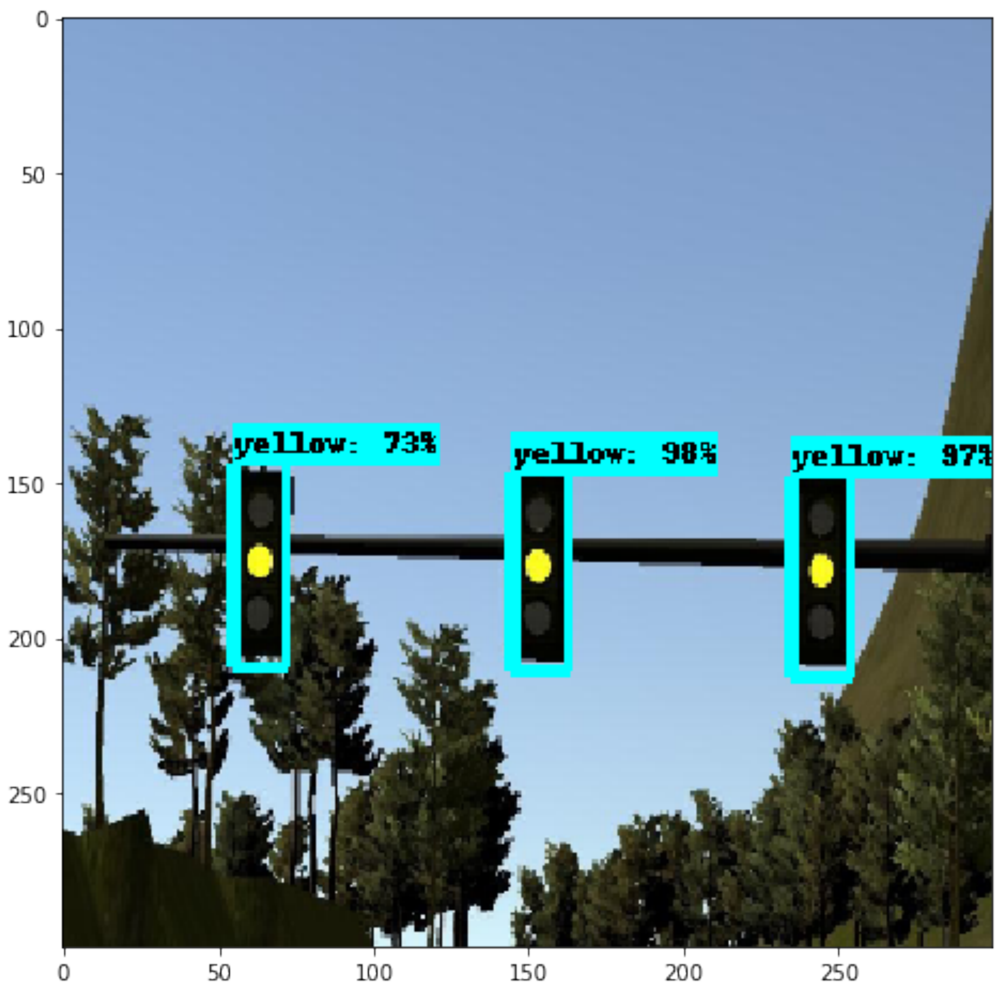
Our traffic light detector was built on top of Google's open source object recognition API [1].

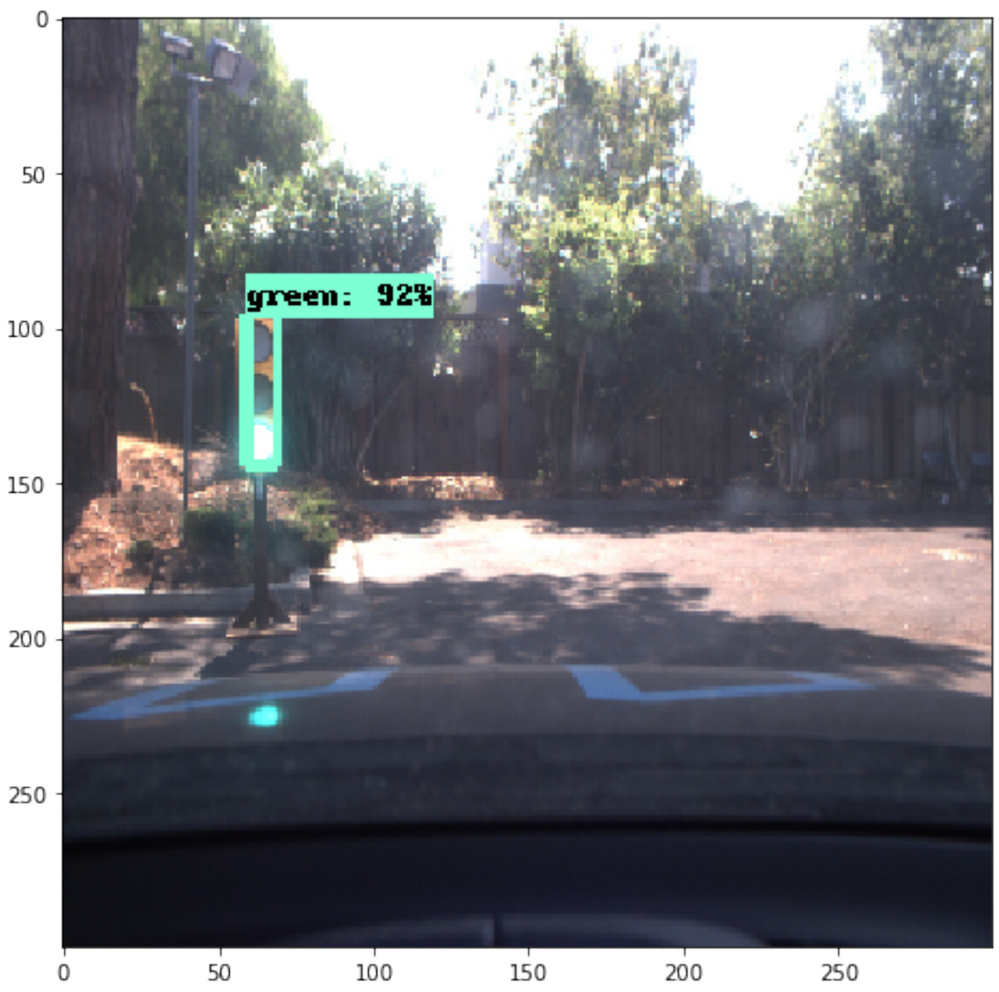
This API was released alongside a very interesting paper [2] that compares the precision and inference time of several different architectures. Since inference time was critical for our application we decided to use a SSD with Mobilenet as feature extractor.

The model was trained using a small dataset with images from both the simulator and the udacity test track. It took a little over 300 epochs (55,000 examples in total) for the model to converge and the validation accuracy was 0.99 mAP@0.5IOU. To increase accuracy and avoid overfitting we relied on some light data augmentation techniques, such as random horizontal flips and random crops.

We have attached below some examples for both the simulator and the bag file:


red: 98%
red: 98%
red: 9
0
50
100
150
200
250
0
50
100
150
200
250


yellow: 73%
yellow: 98%
yellow: 97%
0
50
100
150
200
250
0
50
100
150
200
250



Simulation Video

YouTube Link