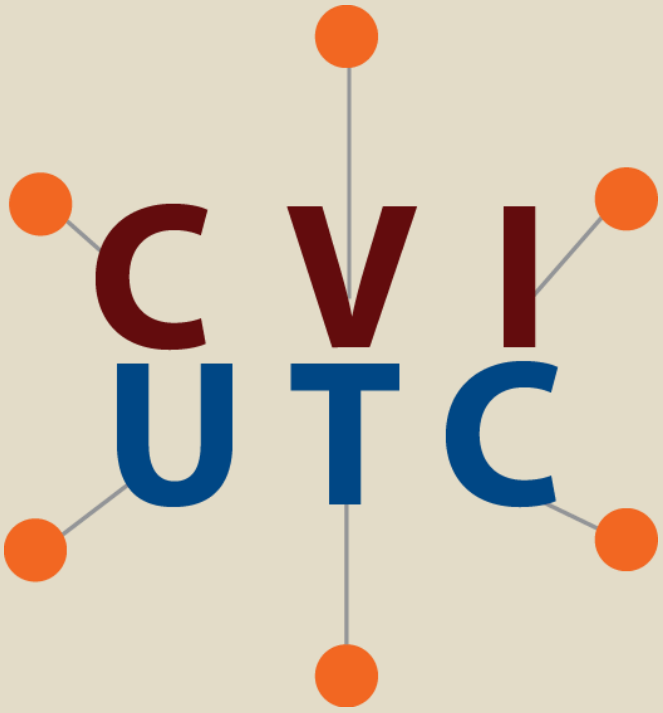


CONNECTED VEHICLE/INFRASTRUCTURE UNIVERSITY TRANSPORTATION CENTER (CVI-UTC)

Field Implementation Feasibility Study of Cumulative Travel-Time Responsive Intersection Control Algorithm under Connected Vehicle Technology

MORGAN STATE UNIVERSITY
VirginiaTech Transportation Institute
UNIVERSITY of VIRGINIA
VDOT

Field Implementation Feasibility Study of Cumulative Travel-Time Responsive Intersection Control Algorithm under Connected Vehicle Technology

Prepared for the Research and Innovative Technology Administration (RITA);
U.S. Department of Transportation (US DOT)

Grant Project Title:
Advanced Operations Focused on Connected Vehicles/Infrastructure (CVI-UTC)

Consortium Members:
Virginia Tech Transportation Institute (VTTI),
University of Virginia (UVA) Center for Transportation Studies,
and Morgan State University (MSU).

Submitted by:

Virginia Tech Transportation Institute
3500 Transportation Research Plaza
Blacksburg, VA 24061

Program Director:
Dr. Thomas Dingus
Program Director, Connected Vehicle/Infrastructure University Transportation Center
Director, Virginia Tech Transportation Institute
Professor, Department of Biomedical Engineering and Mechanics at Virginia Tech
tdingus@vtti.vt.edu
(540) 231–1501

Report Authors:
Saerona Choi, Ph.D.
Research Associate
Department of Civil and Environmental Engineering
University of Virginia
sc2gs@virginia.edu
(434) 924-6347

Byungkyu Brian Park, Ph.D.
Associate Professor
Department of Civil and Environmental Engineering
University of Virginia

Joyoung Lee, Ph.D.
Assistant Professor
Department of Civil and Environmental Engineering
New Jersey Institute of Technology

DUNS: 0031370150000 EIN: 54-6001805

Grant Funding Period: January 2012 – July 2016
Final Research Reports

March 31, 2016

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation's University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

Connected Vehicle/Infrastructure UTC

The mission statement of the Connected Vehicle/Infrastructure University Transportation Center (CVI-UTC) is to conduct research that will advance surface transportation through the application of innovative research and using connected-vehicle and infrastructure technologies to improve safety, state of good repair, economic competitiveness, livable communities, and environmental sustainability.

The goals of the Connected Vehicle/Infrastructure University Transportation Center (CVI-UTC) are:

- Increased understanding and awareness of transportation issues
- Improved body of knowledge
- Improved processes, techniques and skills in addressing transportation issues
- Enlarged pool of trained transportation professionals
- Greater adoption of new technology

Abstract

This project utilized the Connected Vehicle (CV) environment, which provides two-way wireless communications between vehicles and infrastructure, to (1) improve the Cumulative Travel-time Responsive (CTR) Intersection Control Algorithm under low CV market penetration by utilizing Bluetooth technology, and (2) assess potential benefits of the CTR algorithm by examining mobility, energy, and greenhouse emissions measures. The project team developed and evaluated a hardware-in-the-loop simulation to ensure that the developed CTR algorithm will work with an existing traffic controller on the Northern Virginia Connected Vehicle Test Bed.

The team enhanced the CTR algorithm and evaluated its impact to verify the feasibility of field implementation. Two prediction techniques, a standard Kalman filter (SKF) and an adaptive Kalman filter (AKF), were applied to estimate cumulative travel time for each phase in the CTR algorithm. In addition, traffic demand, the market penetration rate (MPR), and the types of available data were also considered in evaluating CTR algorithm performance. The Lee Highway and Nutley Street intersection on the Northern Virginia Connected Vehicle Test Bed was selected for a case study and simulated within VISSIM.

The results showed that the CTR algorithm's performance depends on the MPR, as the information collected from CVs is a key CTR algorithm-enabling factor. However, this study found that the MPR could be relaxed (1) when the data were collected from both CV and infrastructure sensors, and (2) when an AKF was adopted in the CTR algorithm. The minimum MPRs required to outperform the current actuated traffic signal control were empirically found for each prediction technique and types of available data—data from both Connected Vehicle and infrastructure sensors, or Connected Vehicle's data only. Even without the infrastructure sensors, the CTR algorithm could be considered for implementation at an intersection with high traffic demand and a 50% to 60% MPR. As the MPR for this field evaluation was around 14%, much lower than the minimum 20% required with an AKF incorporated, the project team could not implement the proposed algorithm. Instead, the team developed an implementation plan that can be easily adopted by traffic engineers once the MPR reaches 20% or higher.

Acknowledgments

This research was supported by the Connected Vehicles/Infrastructure University Transportation Center. We express our gratitude for the support given by Virginia Department of Transportation traffic engineers Mark Hagan, Ling Li, and Nhan Vu.

Table of Contents

List of Figures

List of Tables

Introduction

According to the January–March 2015 Urban Congestion Report, the average duration of daily congestion in the United States—the extra time lost due to the difference between congested speed and free-flow speed—was approximately 5 hours [1]. To deal with congestion in urban areas, traffic engineers and researchers have developed various adaptive traffic signal control systems. These systems collect vehicle information in real time to optimize signal timing plans by changing the length and sequence of the phases to serve current traffic demands. There are a number of widely used systems (Table 1), including the Split Cycle Offset Optimization Technique (SCOOT), Sydney Coordinated Adaptive Traffic System (SCATS), Real Time Hierarchical Optimized Distributed Effective System (RHODES), ACS-Lite, Optimization Policies for Adaptive Control (OPAC), and InSync [2].

Table 1. Descriptions of Current Adaptive Traffic Signal Control Systems [3]

System	Year and country developed		Goal	Methodologies
SCOOT	1970	UK	Minimizes delay with relative importance on stop	– Optimizes splits – Cycle and offsets – Real-time optimization of signal timing
SCATS	1970	Australia	Minimizes stops, delay (heavy traffic), and travel time	– Optimizes splits – Cycle and offsets – Selects from a library of stored signal timing plans
RHODES	1990	USA	Proactively responds to the natural stochastic behavior of traffic flow	– Mainly for diamond interchange locations
OPAC	1990	USA	Minimizes delay and stops over a pre-specified horizon	– The network is divided into independent sub-networks
ACS Lite	1990-2006	USA	Adjusts splits and offsets on a cycle-by-cycle basis	– Operates with predetermined coordinated timing plans – Automatically adjust splits and offsets accordingly
InSync	2008	USA	Services movement stages to minimize queues and delays	– Uses queue lengths, volumes and occupancy to optimize time tunnels

Most adaptive traffic control strategies for urban networks that were developed to deal with traffic congestion face two big challenges. First, since these systems rely mostly on prediction techniques based on approaching demand, vehicle arrival patterns, and turning movement rates, any inaccuracy in the prediction technique undermines the performance of traffic control systems. To overcome inaccuracies and improve system performance, several researchers have applied advanced prediction techniques to traffic control algorithms. For real-time travel time prediction problems, Kalman-filter-based algorithms and time-series models have received great attention among parametric models, and have been compared with other methods. Several researchers have employed an advanced Kalman filter to overcome the limitation of the Kalman filter that Gaussian noise might not be consistent in field data. These approaches include use of an extended Kalman filter [4]–[6], an adaptive Kalman filter [7], [8], and an unscented Kalman filter [9]. In addition, a neural network model, which is a nonparametric prediction model, has been used due to its well-

known learning and pattern-recognition abilities [10]–[14]. Currently, the k-nearest neighbors approach is widely used as a non-parametric, short-term prediction method, and it can be easily extended to handle a multivariate problem using historical data or real-time data [15].

Second, real-time data for adaptive control systems are collected from infrastructure-based sensors, such as video cameras or loop detectors, that are fixed-point sensors. However, unreliable prediction of vehicle locations and speeds can lead to suboptimal control. Moreover, travel times cannot be collected directly until vehicles completely pass the sensors. Hence, travel times need to be estimated by using an algorithm. Under a vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication environment, referred to collectively as V2X, connected vehicles (CVs) can send their trajectories to other vehicles and infrastructure through communication-based devices in real time, and the intersection control algorithm can use directly measured travel time data.

Recently, many researchers have investigated how to take advantage of communication-based traffic data to improve operational efficiency and traffic safety. Several methodologies and algorithms have been proposed to allow vehicles to cross safely at an intersection under a V2X communication environment. Such algorithms were developed to coordinate individual vehicles' maneuvers using predicted trajectories or calculated crash potential so that vehicles can safely cross the intersection [16], [17] Guler et al. [18] proposed an algorithm that incorporates information from CVs to determine the sequence of departures from an intersection, and developed an algorithm to evaluate the impacts of autonomous vehicle control and detailed vehicle information. Dujardin et al. [19] proposed a multi-objective optimization interactive procedure that considers total waiting time and the number of stops based on an adaptive optimization system. Feng et al. [20] proposed an algorithm to optimize the phase sequence and duration by solving a two-level optimization problem: minimization of total vehicle delay and minimization of queue length under a V2X environment. Their traffic control algorithms using communication-based data worked well compared with current adaptive signal control systems when a 100% market penetration rate (MPR) was assumed, but the performance significantly dropped as MPR decreased.

Concept of the CTR Algorithm

The Cumulative Travel-time Responsive (CTR) algorithm is a real-time intersection control strategy. As shown in Figure 1, the CTR algorithm determines the optimal green split for the next time interval by identifying the maximum cumulative travel time (CTT) measured by both CV and infrastructure-based sensors under a V2X communication environment. CTT is defined as the sum of the elapsed time spent by individual vehicles for each phase at an intersection. Employed as a real-time measurement for the CTR algorithm, CTT enables the capture of instantaneous delays caused by queues and waiting time at an intersection. Given this information, the CTR algorithm can respond rapidly to a congested traffic condition to reduce the delay and total travel time of the intersection.

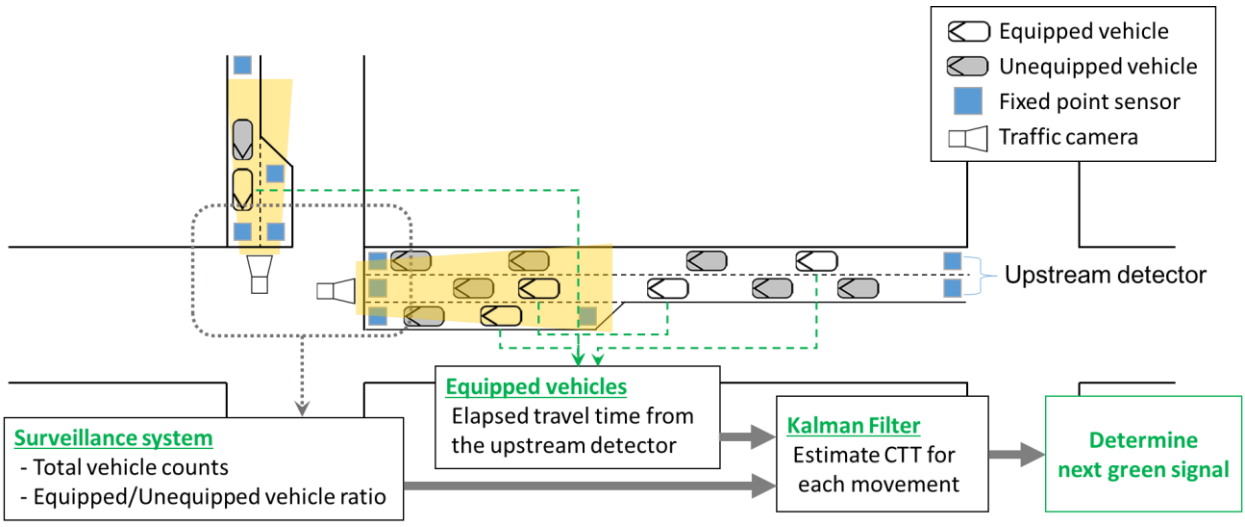


Figure 1. Concept of the CTR algorithm.

Figure 2 depicts the CTR algorithm. The travel time data of individual vehicles equipped with a communications device is collected to implement the CTR algorithm. Subsequently, the CTT for each phase is calculated. The phase with the longest CTT is compared with the current green time phase and the CTR algorithm determines whether the current timing for the green phase should be kept or not.

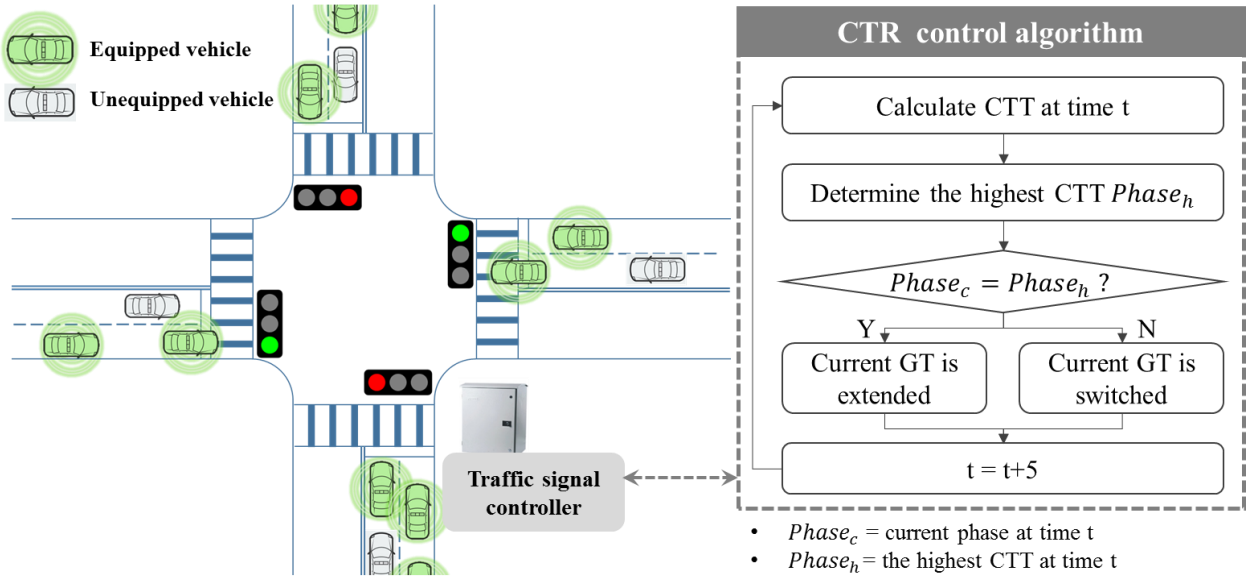


Figure 2. Details of the CTR algorithm.

Research Objectives

The objectives of this research were as follows:

1. To analyze the effectiveness of a CTR algorithm by incorporating MPRs, traffic demand, and types of available data (i.e., data from both CV and infrastructure sensors vs. CV data only).

2. To verify the feasibility of field implementation in the near future considering an adaptive Kalman filter (AKF) algorithm to improve prediction performance under variable MPRs.

3. To evaluate the CTR algorithm using a calibrated VISSIM simulation environment compared with a current traffic signal control algorithm based on infrastructure sensors by considering MPRs in terms of mobility and environmental sustainability.

Methods

Taking into consideration a variety of MPRs, this research took the following steps to evaluate the performance of the CTR algorithm in comparison with the current traffic signal control system.

1. Various Kalman filter algorithms were evaluated and selected.

2. An aparatus was designed for potential field implementation and simulation consideration.

3. A study area was selected and a simulation environment was established using VISSIM [21], a microscopic simulation package.

4. Field data (i.e., traffic volume, signal timing plans) were collected during both peak and off-peak hours for VISSIM model calibrations.

5. In the simulation environment, real-time CTTs were collected from calibrated VISSIM models and estimated by Kalman filter algorithms under imperfect MPR conditions. It is worth noting that an AKF was employed to improve the prediction performance as the AKF could dynamically adjust coefficients for the system and observation noise under the congested situation. The project team also considered two cases of available data:

    - **Case 1: "CV and Infra."** Data are obtained from both CVs (e.g., travel time) and infrastructure sensors (e.g., total number of vehicles).

    - **Case 2: "CV Only."** Data are obtained from the CVs only.

6. The effectiveness of the CTR algorithm was evaluated by comparing its results with the actuated traffic signal control under various values of MPR in terms of mobility and environmental sustainability. The selected performance measurements included travel time, average speed, throughput, delay, CO_2 emissions, and fuel consumption.

Kalman Filter Algorithms

The CTT is a key factor when the CTR algorithm determines the next signal phase. In other words, the performance of the CTR algorithm depends on the accuracy of the CTT. As measurement of the CTT depends on MPR, a low MPR would likely undermine the CTR algorithm's performance. To help solve this problem, an advanced prediction technique can be employed to improve the estimation accuracy of the CTTs. To this end, this research applied Kalman filter algorithms to compensate for imperfect market penetration.

The Kalman filter technique has been widely implemented to estimate future traffic conditions using collected data [22]–[24]. This method relies on stochastic and dynamic models that describe the behavior of the state-space vector and the relationship between the state space and the measurement vector. The algorithm works by using a two-step process that involves a time update and a measurement update. In the first step, the algorithm estimates the current state variables, along with their uncertainties. Once the outcome of the next measurement is observed, these estimates are updated using a weighted average in the second step, with more weight being given to estimates with higher certainty. In addition, this algorithm can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix because of the algorithm's recursive nature.

The state-space equation in Equation (1) explains the current state (x_k) that is the result of the previous state (x_{k-1}), the previous input action (u_{k-1}), and the noise from the previous time step. The measurement equation presented in Equation (2) explains the current measurement (z_k) that results from the current estimated states with noise. w_k and v_k are process noise and measurement noise with variance of Q and R, and are assumed to have a Gaussian noise distribution. The observation matrix, *H*, in Equation (2), is employed to adjust the difference between the measured states (the collected CTTs from CVs) and the predicted states (the obtained CTTs from the state-space equation). If MPR is 100%, the observation matrix should be an identity matrix.

- State-space equation: $x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$, (1)

  where *A* is the transition matrix for state mapping, *B* is the transition matrix for input mapping, and $w_k \sim N(0, Q)$.

- Measurement equation: $z_k = Hx_k + v_k$, (2)

  where *H* is the observation matrix and $v_k \sim N(0, R)$.

The transition matrices A and B in Equation (1) are employed to account for the relationship between control activities and the results. These matrices need to be determined by considering road and traffic characteristics such as geometric condition (i.e., the number of lanes for each phase), volume (i.e., the number of approaching vehicles), and signal status because the cumulative travel time could be affected by various external activities. Hence, the project team utilized

Equation (3), which comes from a previous study [25]. As noted, when MPR is 100%, the observation matrix becomes an identity matrix. If infrastructure sensors are not installed (e.g., loop detectors), the total number of vehicles (q_i in Equation [3]) is not available. In this case, this equation is modified as Equation (4).

- Case 1: CV and Infra

$$t_{i,k} = \alpha t_{i,k-1} + \beta q_{i,k-1} + \mu g_{i,k-1} + \sigma NL_i \quad (3)$$
$$t_{j,k} = \gamma t_{j,k-1} + \delta t_{i',k-1} + \varepsilon q_{j,k-1} + \tau g_{j,k-1}$$

- Case 2: CV Only

$$t_{i,k} = \alpha t_{i,k-1} + \mu g_{i,k-1} + \sigma NL_i \quad (4)$$
$$t_{j,k} = \gamma t_{j,k-1} + \delta t_{i',k-1} + \tau g_{j,k-1}$$

where:

t_k: cumulative travel time at time interval k
$q_{i,k-1}$: vehicle counts of phase i at $k-1$
$g_{i,k-1}$: length of green time of phase i at $k-1$
NL_i: the number of lanes of phase i
i, j: the number of phases for through traffic and left turns based on the National Electrical Manufacturers Association (NEMA) standard, respectively
i': the number of through traffic phases corresponding to left-turn traffic
j: the number of phases for left turns
$\alpha, \beta, \gamma, \delta, \varepsilon, \mu, \tau, \sigma$: coefficients.

The covariance matrices in the standard Kalman filter (SKF) can be estimated using Minimum Norm Quadratic Unbiased Estimation (MINQUE) [26]. However, MINQUE is an offline tuning process that is not suitable for real-time implementation. The estimation of the noise variance is very important in order to correctly tune the filter because it determines the Kalman gain. In this project, an AKF was considered to address this issue. The basic idea of the AKF is to update the covariance matrices at every time interval by using a covariance matching technique called multiple model adaptive estimation (MMAE) [27] to reduce uncertainty in the error of covariance. Generally, the procedure for the AKF is as follows:

1. The state propagation and prior state estimation error covariance are estimated.
2. Observation errors are computed.
3. The observation process covariance matrix is updated.
4. The Kalman gain is calculated.
5. The posterior state estimation and posterior state estimation error covariance are estimated.

6. State estimation errors are computed.

7. The state process covariance matrix is updated.

More details about the Kalman filter algorithms are available in previous studies [7], [8].

Study Apparatus

The project team designed an apparatus for eventual field implementation; however, it is important to note that the current project did not progress to a full field implementation. Therefore, the conceptual apparatus designed was used for bench testing and simulations of a fully-implemented CTR algorithm in a V2X environment under various MPRs.

Bluetooth Technology as a CV Surrogate

Given that CV devices have not been widely deployed, the project team utilized Bluetooth technology in the travel time estimation. Figure 3 shows the set of components that were utilized to supplement the low market penetration of CV devices.

The travel time of each turning movement at an intersection was captured using Bluetooth technology. This required position information to be acquired from each Bluetooth device. There have been several research efforts to identify the position of Bluetooth devices to accommodate location-based services under the Bluetooth environment [28], [29]. In this research project, we used a Received Signal Strength Indicator (RSSI)-based Bluetooth positioning method [29]. In a line-of-sight (LOS) scenario, the most dominant factor affecting the strength of the radio signal is distance between a sender and a receiver. Since a Type 2 Bluetooth reader has a communication radius of 10 to 20 meters, it operates in an LOS condition when approaching an intersection. Thus, analyzing RSSI will most likely allow the distance to the Bluetooth device to be captured precisely, as shown in Figure 3. By utilizing this distance information, the project team developed an algorithm to estimate the CTT of each turning movement. The Zigbee module, shown inFigure 3, wirelessly sent the data collected from the Bluetooth reader to roadside equipment (RSE).

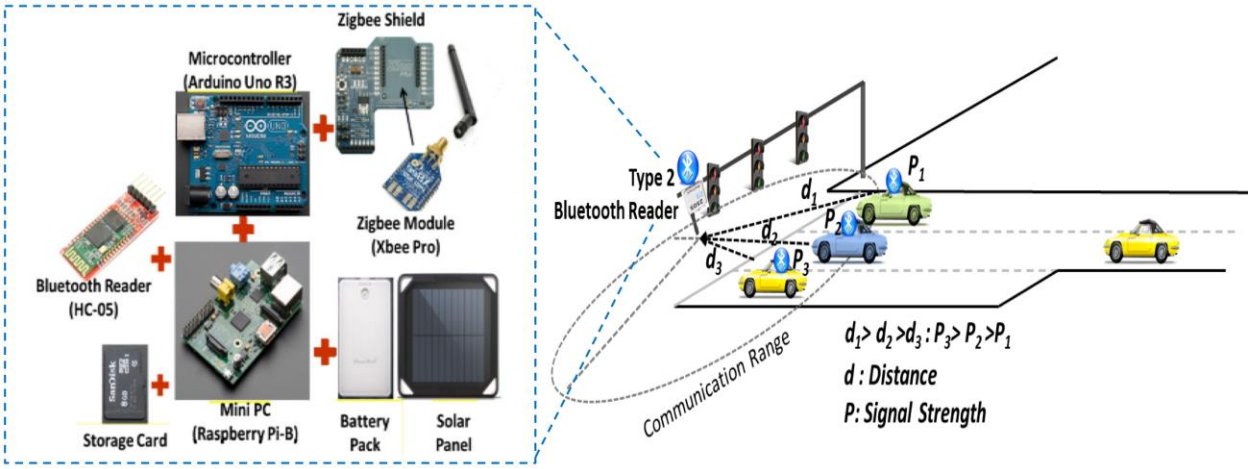


Figure 3. Conceptual illustration of RSSI-based distance measurement for Bluetooth devices and sensor unit components.

Hardware-in-the-Loop Simulation (HILS) of CTR Algorithm in Traffic Controller

When a new traffic signal control logic or a new or updated traffic controller is to be deployed in the field, traffic engineers can utilize hardware-in-the-loop simulation (HILS). The project team developed a HILS [30] environment to operate the CTR algorithm. The HILS consisted of a 2070L traffic signal controller, controller interface device (CID), a Bluetooth reader developed by Lee et al. [31], a communication device between the server and Bluetooth reader, and VISSIM.

Figure 4 illustrates the HILS configuration for analyzing the CTR algorithm under a CV environment. First, Bluetooth readers capture the Medium Access Control (MAC) addresses of Bluetooth devices in approaching individual vehicles in each direction every 5 seconds. Second, the collected MAC addresses are transmitted from Bluetooth readers to a remote server through Zigbee-based short-range communications [31]. The MAC address data are stored in a database in the server computer. Third, a program on the server matches the MAC addresses from downstream and upstream for each direction and computes the travel time of equipped vehicles in real time. Fourth, using these travel times, the CTR algorithm determines the next green phase timing and sends this information to the CID. Fifth, the CID converts digital signals to analog signals, and this signal is sent to controller hardware. In addition, the controller sends the signal information to the signal head. Since Step 5 is not available for indoor experiments, the project team only implemented Steps 1 through 4 when analyzing the CTR algorithm in the HILS configuration.

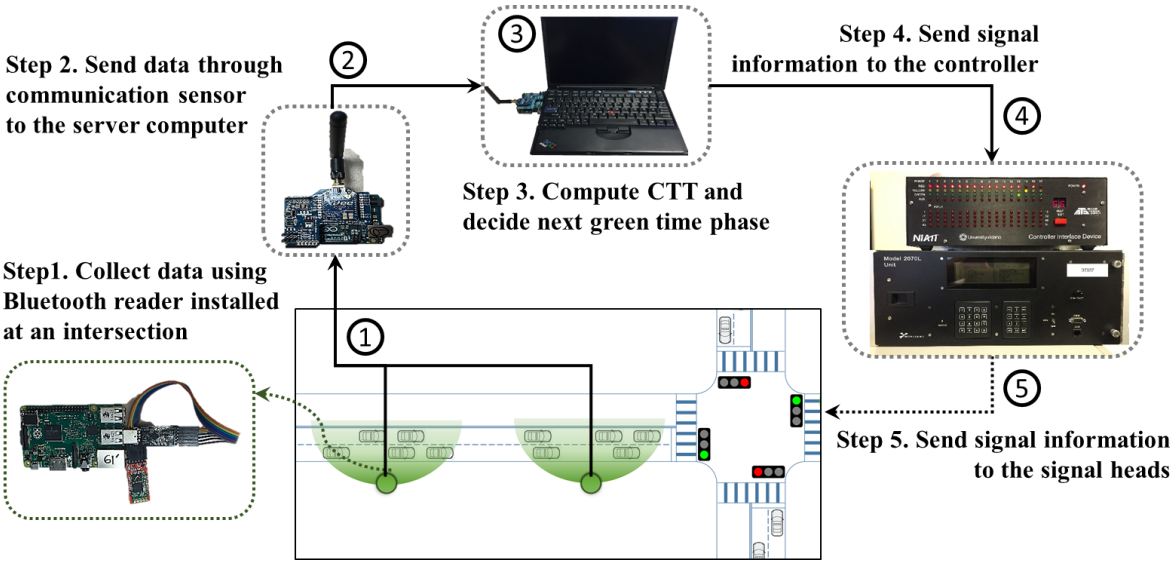


Figure 4. HILS configuration for CTR algorithm.

VISSIM was used to establish a simulation-based environment for analysis of the CTR algorithm. In addition, a C# programming language on VISSIM's COM interface, which allows additional external control of a simulation model, was used to implement the CTR algorithm. Figure 5 describes the simulation-based analysis procedure in this research using the VISSIM COM interface for the CTR algorithm. At time interval *t*, VISSIM collects elapsed time information for equipped vehicles as travel time measurements for each phase (e.g., phase numbers 1, 3, 5 and 7 in the NEMA standard) and sends the information to the CTR algorithm. If MPR is 100%, the CTR algorithm immediately calculates CTTs. If MPR is imperfect, CTTs are estimated from Kalman filter algorithms, either SKF or AKF. To calculate the matrices in the Kalman filter algorithms, this study used the dynamic linked library in MATLAB. Once the largest CTT phase is determined as next green phase using the estimated CTTs, the current green signal is either extended or switched to the largest CTT phase by the CTR algorithm.

In the actual implementation of the CTR algorithm, it would be crucial to apply a suitable update interval to evaluate the CTT of each phase and determine the next green phase. The previous simulation-based evaluation research [22] on the CTR algorithm used a 5-second update interval. In this research, the impact of update intervals was investigated using various intervals (i.e., 4, 5, 6, and 7 seconds). It turned out that the results under these intervals were not statistically significantly different. Thus, the CTR algorithm under 5 seconds update interval was used throughout this reserach.

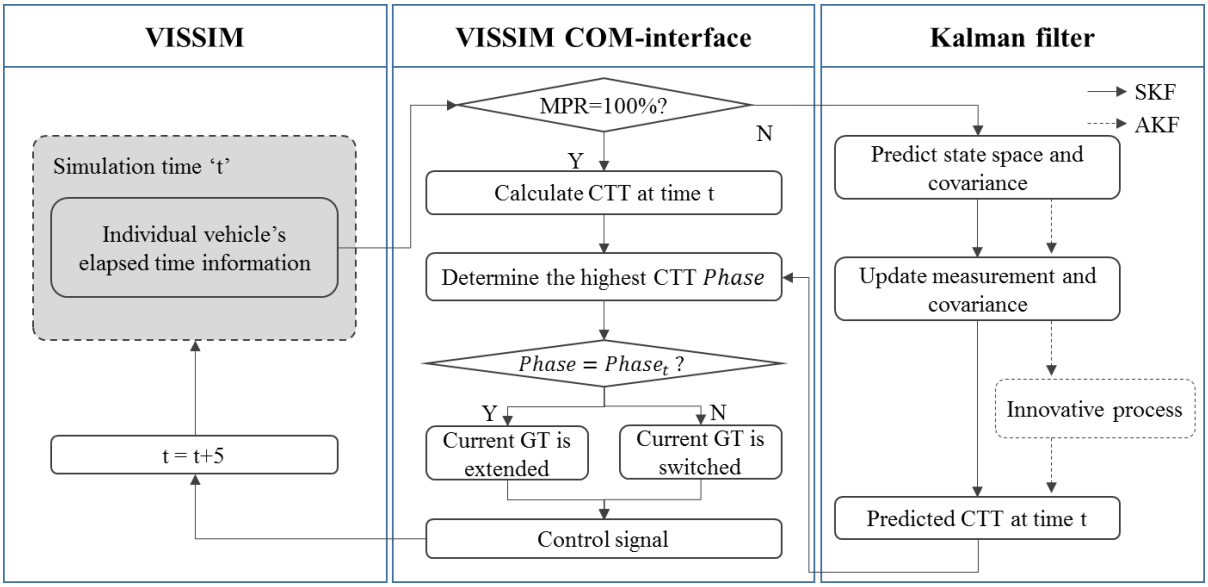


Figure 5. Simulation-based analysis procedure.

Sensor Configuration for Collecting Travel Time Data

Figure 6 shows the configuration of devices used to collect travel time data from Bluetooth signals. To configure the Raspberry Pi and sensors, we installed the Rasbian operating system on the Raspberry Pi. For efficiency, we used Putty (www.putty.org)—an open source client supporting a Secure Shell (SSH) connection—to run Terminal (Linux command) in the Raspberry Pi at a laptop computer. Then, we connected a Bluetooth sensor and communication device to the Raspberry Pi. Before we used the sensor combination, the Bluetooth reader needed to be configured for time interval and the number of MAC addresses that the Bluetooth reader could read during the time interval. For these, several Python scripts developed in this project were copied to the Raspberry Pi. "*Serialtest.py*" was used to set the Bluetooth reader, and "*MultiBT.py"* was used to collect Bluetooth MAC addresses and save them on the Raspberry Pi.

To configure communication devices, we combined the Zigbee shield and Zigbee module with an Arduino that can connect the central processing unit (CPU) board to a variety of interchangeable add-on modules known as shields. We set Arduino and Zigbee as follows: i) The Arduino website (www.arduino.cc) provides a basic installation program for Arduino with Zigbee, and ii) the DIGI website (www.digi.com) supports XCTU software to set up the Zigbee module as a coordinator or router. To maximize the scanning range of the sensor considering vehicles' height, we recommend a height of approximately 5 to 6 feet for installation of the sensor.

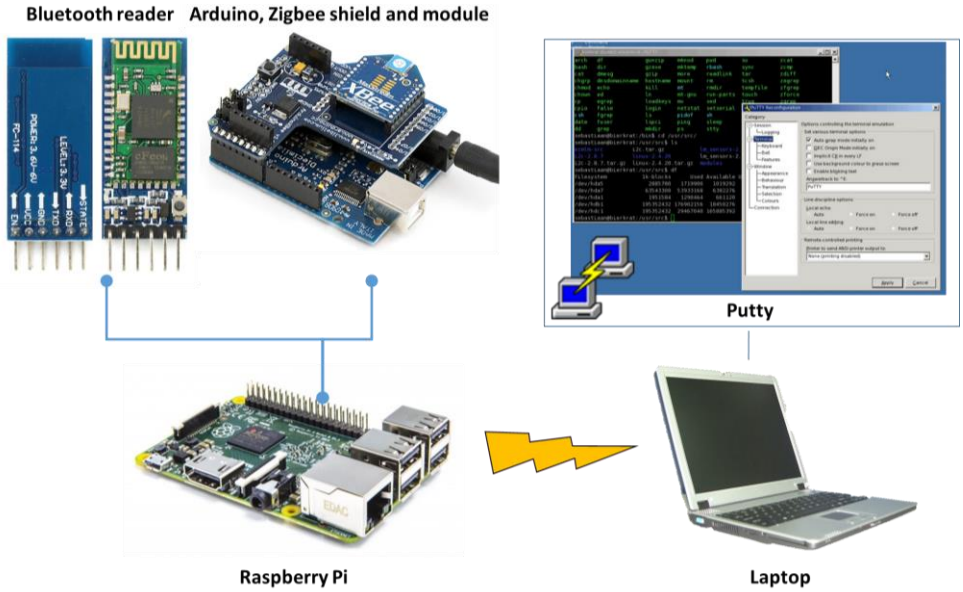


Figure 6. Configuration of devices.

Simulation

Study Area Calibration

Once the HILS results indicated that the CTR algorithm worked as expected and the results were promising, the team presented the findings to Virginia Department of Transportation (VDOT) traffic engineers and requested permission to deploy the CTR algorithm at the test site intersection. The Lee Highway and Nutley Street intersection on the Northern Virginia Connected Vehicle Test Bed [32] was selected, as shown in Figure 7. The intersection operates according to actuated signal control. Nutley Street connects to Interstate 66 as well as Lee Highway, and there are high inbound traffic volumes during peak hours. To establish and calibrate a simulation environment of the study area, field data (e.g., traffic volume, geometrical characteristics, and signal timing plans) were collected during a peak hour (7 a.m. to 8 a.m.) and an off-peak hour (3 p.m. to 4 p.m.). The eastbound and westbound approaches have permitted exclusive left-turn signals; the southbound and northbound approaches have protected through-left-turn signals.

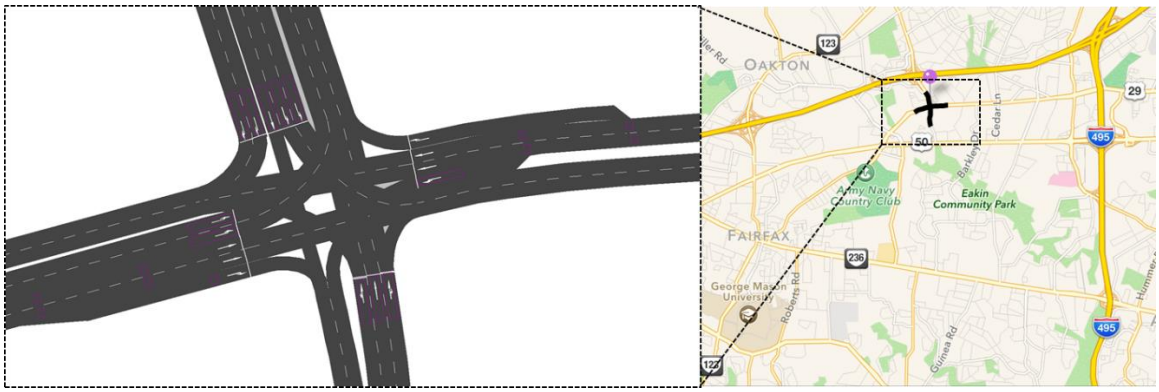
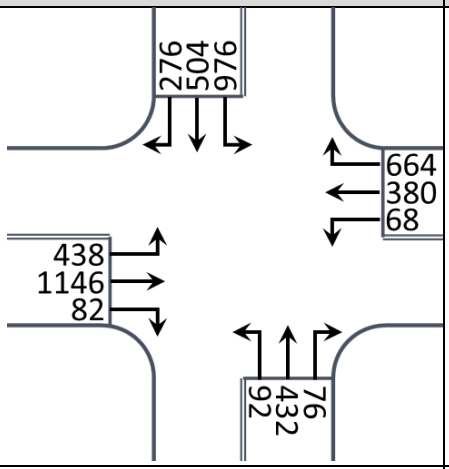
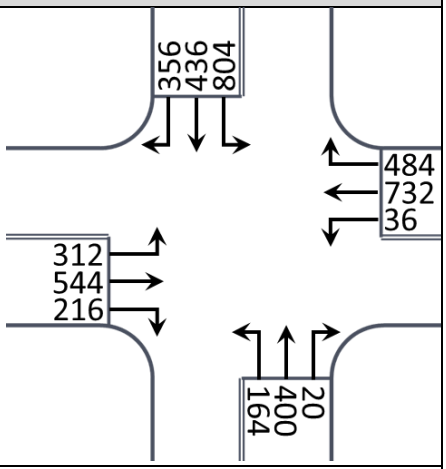


Figure 7. VISSIM network for study area [33].

Two sets of traffic volume data were collected in the study area during peak and off-peak hours. In both, higher traffic volume rates were found at Lee Highway (i.e., east-west directions) than at Nutley Street. The left-turn traffic in the southbound direction and the right-turn traffic in the westbound direction had higher volume rates than through traffic due to the traffic volume going to and from I-66. The VISSIM simulation environment was calibrated and compared in terms of using total travel time and average speed by mean absolute percentage error (MAPE), as described in Table 2. The MAPEs ranged from 5% to 15%. Measures of effectiveness (MOEs) regarding operational efficiency and environmental sustainability were analyzed across 10 replications to assess the performance of the CTR algorithm.

Table 2. Simulation Results Using Existing Traffic Signal Timing and Traffic Volume

Parameters	Peak hour	Off-peak hour
Volume (vehicles/h)		
Total travel time (h)	169.089 (MAPE 5%)	98.659 (MAPE 13%)
Average speed (mph)	11.040 (MAPE 15%)	16.697 (MAPE 8%)
Delay (s)	92.310	48.214
CO_2 (kg/unit)	0.787	0.473
Fuel (kg/unit)	0.615	0.315

Model Estimation for Kalman Filter Algorithms in the CTR Algorithm

In the state-space equation, coefficients were estimated considering external traffic characteristics such as the number of lanes, the existence of a left-turn bay, and signal status. These estimated coefficients should be statistically significant because they affect the accuracy of estimated CTTs. To estimate coefficients by a regression model, the project team collected 2,880 data records, including CTTs, the number of vehicles, and length of green time from the calibrated VISSIM simulation and used SPSS 22, which is a statistical analysis package. Considering data availability, two types of state-space equations were developed, as shown in Table 3: one set for the case when both CV and infrastructure sensor data are available (Case 1: "CV and Infra") and a second set for the case when only CV data are available (Case 2: "CV Only"). All parameters are statistically significant with a 95% significance level for both equations, and R^2 values, which represent model performance, are close to 1.0. Using coefficients for each equation, Equations (3) and (4) can be written as Equations (5) and (6), respectively.

- Case 1: CV and Infra

$$t_{i,k} = 0.85 \cdot t_{i,k-1} + 3.33 \cdot q_{i,k-1} - 22.90 \cdot g_{i,k-1} + 8.13 \cdot NL_i \quad (5)$$
$$t_{j,k} = 0.92 \cdot t_{j,k-1} - 0.01 \cdot t_{i',k-1} + 4.11 \cdot q_{j,k-1} - 22.48 \cdot g_{j,k-1}$$

- Case 2: CV Only

$$t_{i,k} = 0.92 \cdot t_{i,k-1} - 22.81 \cdot g_{i,k-1} + 13.68 \cdot NL_i \quad (6)$$
$$t_{j,k} = 0.98 \cdot t_{j,k-1} + 0.02 \cdot t_{i',k-1} - 19.06 \cdot g_{j,k-1}$$

Table 3. Regression Model to Estimate Coefficients in Kalman Filter Using Simulated Data

Scenario	Equations	Model Summary					Performance		
		Parameter	B	Std. Error	t	Sig.	R	R^2	Adjusted R^2
CV and Infra	Equation for THRU	α	0.85	0.01	67.81	0.00	0.976	0.953	0.953
		β	3.33	0.49	6.72	0.00			
		μ	-22.90	0.89	-25.81	0.00			
		σ	8.13	1.12	7.23	0.00			
	Equation for LT	γ	0.92	0.01	109.61	0.00	0.975	0.951	0.951
		δ	-0.01	0.01	-2.20	0.03			
		ε	4.11	0.28	14.83	0.00			
		τ	-22.48	0.74	-30.41	0.00			
CV Only	Equation for THRU	α	0.92	0.01	167.64	0.00	0.976	0.952	0.952
		μ	-22.81	0.89	-25.52	0.00			
		σ	13.68	0.77	17.79	0.00			
		γ	0.98	0.01	132.81	0.00	0.973	0.947	0.947

Scenario	Equations	Model Summary					Performance		
		Parameter	B	Std. Error	t	Sig.	R	R^2	Adjusted R^2
	Equation for LT	δ	0.02	0.01	2.78	0.01			
		τ	-19.06	0.73	-26.16	0.00			

NOTE: THRU is through; LT is left turn.

Scenarios and MOEs

Eleven different MPR values were applied to the simulation scenarios to evaluate the CTR algorithm. The MPRs ranged from 0% (current signal control) to 100% (perfect CV environment) and were incremented by 10%. The team used two sets of traffic volume data, including a peak hour and an off-peak hour. In addition, two types of communication techniques and two types of Kalman filters were considered. Thus, a total of 82 scenarios were developed to evaluate the CTR algorithm, and five replications were made for each scenario. For comparison purposes, the team employed the following MOEs: total travel time (h), average speed (mph), and delay (s) as mobility measures, and the amount of CO_2 emissions per vehicle and fuel consumption as environmental sustainability measures. In addition, the VT-Micro model [34] was employed to estimate the emissions and fuel consumption for each scenario using speed and acceleration in vehicle trajectory data collected by the VISSIM simulation.

Table 4. Analysis Scenarios

Traffic signal control	MPR (%)	Scenario number			
		Peak hour		Off-peak hour	
		CV and Infra	CV Only	CV and Infra	CV Only
Actuated signal control	-	1	-	42	-
CTR algorithm with SKF	10	2	22	43	63
	20	3	23	44	64
	30	4	24	45	65
	40	5	25	46	66
	50	6	26	47	67
	60	7	27	48	68
	70	8	28	49	69
	80	9	29	50	70
	90	10	30	51	71
	100	11	31	52	72
CTR algorithm with AKF	10	12	32	53	73
	20	13	33	54	74
	30	14	34	55	75
	40	15	35	56	76
	50	16	36	57	77

Traffic signal control	MPR (%)	Scenario number			
		Peak hour		Off-peak hour	
		CV and Infra	CV Only	CV and Infra	CV Only
	60	17	37	58	78
	70	18	38	59	79
	80	19	39	60	80
	90	20	40	61	81
	100	21	41	62	82

Results

Feasibility of the CTR Algorithm

Through simulation, the effectiveness of the CTR algorithm compared with the actuated signal control algorithm was evaluated and results are shown in Figure 8 through Figure 12 in terms of MPRs, volume scenarios, communication types, and Kalman filter algorithms. The existing actuated signal control is considered to be up-to-date, as the Northern Virginia traffic engineers have maintained the timing plans well in the area.

Generally, the CTR algorithm's performance improved as the rate of CV-equipped vehicles increased. With 100% MPR under a V2X communication environment, the CTR algorithm significantly improved mobility when compared to the actuated signal control at peak hour; total travel time decreased by 45%–47%, average speed increased by 96%–101%, and delay decreased by 71%–73%. Moreover, at the off-peak hour, travel time decreased by 37%–42%, average speed increased by 57%–70%, and delay decreased by 61%–69%. In terms of environmental sustainability, CO_2 emissions increased by 1%–2% and fuel consumption decreased by 3%–6%; however, these findings were not significant.

Effectiveness of Prediction Technique

An interesting finding is that the CTR algorithm showed different performance by type of Kalman filter algorithm under low MPR conditions. At the off-peak hour, the performance of the CTR algorithm with SKF (represented by rectangle marks in Figure 8 through Figure 12) was about 5%–10% better than that with the AKF (represented by circle marks in Figure 8 through Figure 12). Moreover, the minimum required MPR of the SKF (10%) was lower than that of the AKF (20%). However, the AKF's results were better than the SKF's results at peak hour. In addition, the minimum required MPR of the AKF (20%) was lower than the SKF (30%). Therefore, to guarantee the CTR algorithm's performance for both traffic demands, 30% and 20% MPR would be needed for the SKF and the AKF, respectively. Because the AKF showed better performance than the SKF under imperfect MPR, the AKF is recommended as a prediction technique for the CTR algorithm.

Effectiveness of Data Availability

If information from connected infrastructure is not available, a high CV MPR should be ensured for effective operation of the CTR algorithm. According to the comparison results between the "CV and Infra" (solid line in the figures below) and "CV Only" (dotted or dashed line in the figures below) cases, the minimum required MPRs for the CTR algorithm were 50%–60% at the peak hour and 90% at the off-peak hour to outperform the current actuated traffic signal control. Even at the same MPR for the peak and off-peak hours, the results of the CTR algorithm's performance are different. This is because the quality of information for operating the CTR algorithm is influenced by the number of equipped vehicles. Therefore, infrastructure sensor data are needed for stable algorithm performance. On the other hand, even if there are no infrastructure sensors at the intersection, the CTR algorithm could be considered where high traffic demand is found with 60% MPR. Furthermore, the CTR algorithm could improve mobility over actuated traffic signal control even if MPR is under 50% when the AKF is used.

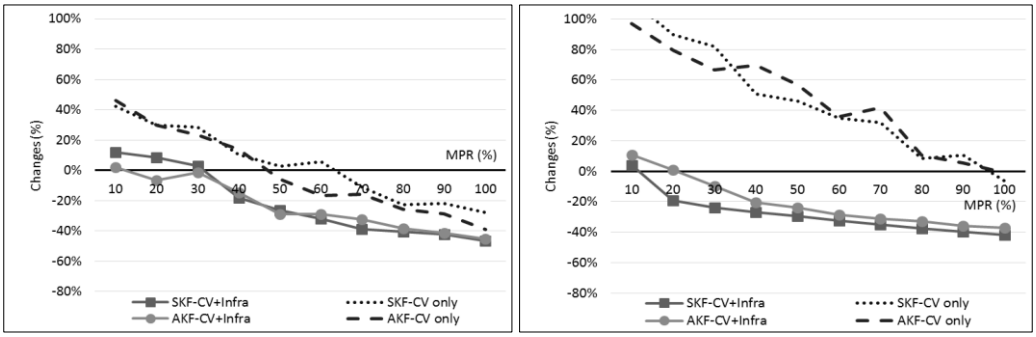


Figure 8. Total travel time (h): peak hour (left), off-peak hour (right).

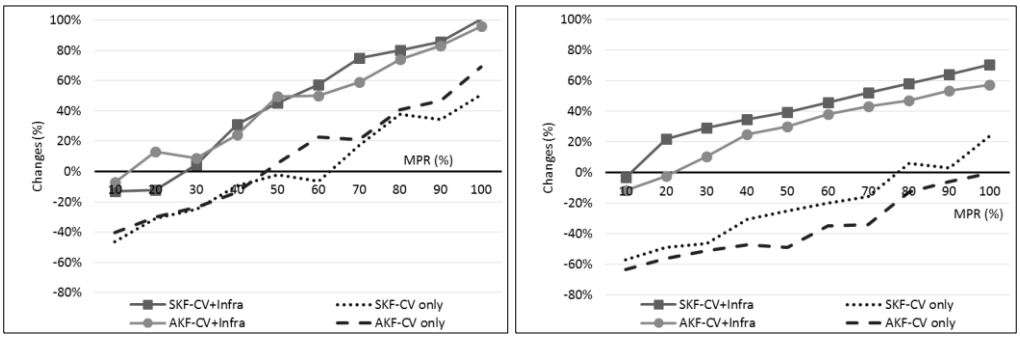


Figure 9. Average speed (mph): peak hour (left), off-peak hour (right).

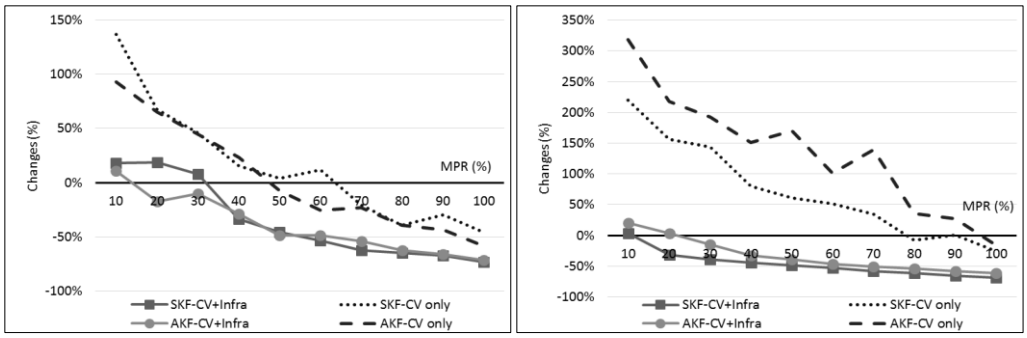


Figure 10. Delay (s): peak hour (left), off-peak hour (right).

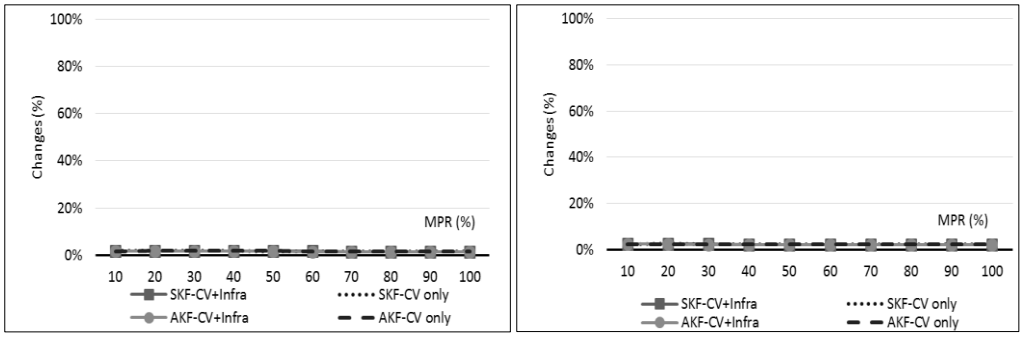


Figure 11. CO_2 (kg/unit): peak hour (left), off-peak hour (right).

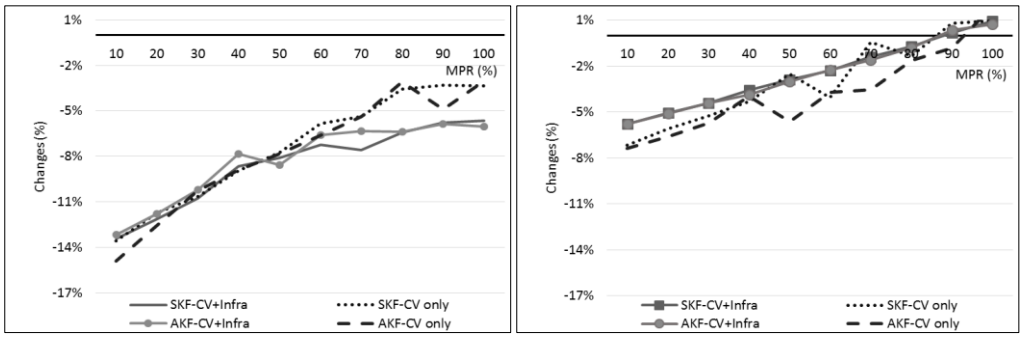


Figure 12. Fuel consumption (kg/unit): peak hour (left), off-peak hour (right).

MPR at the Study Area

The project team investigated the MPR—the percentage of vehicles that have Bluetooth devices—at the study area. We installed eight Bluetooth readers, four communication devices, and a video camera as shown in Figure 13. From 6:30 p.m. to 7:30 p.m. on September 2, 2015, we video recorded the number of vehicles that passed the intersection for both eastbound and westbound traffic as the Bluetooth readers collected Bluetooth MAC addresses. To maximize the number of Bluetooth MAC addresses collected, we installed the devices at a height of 5 feet.

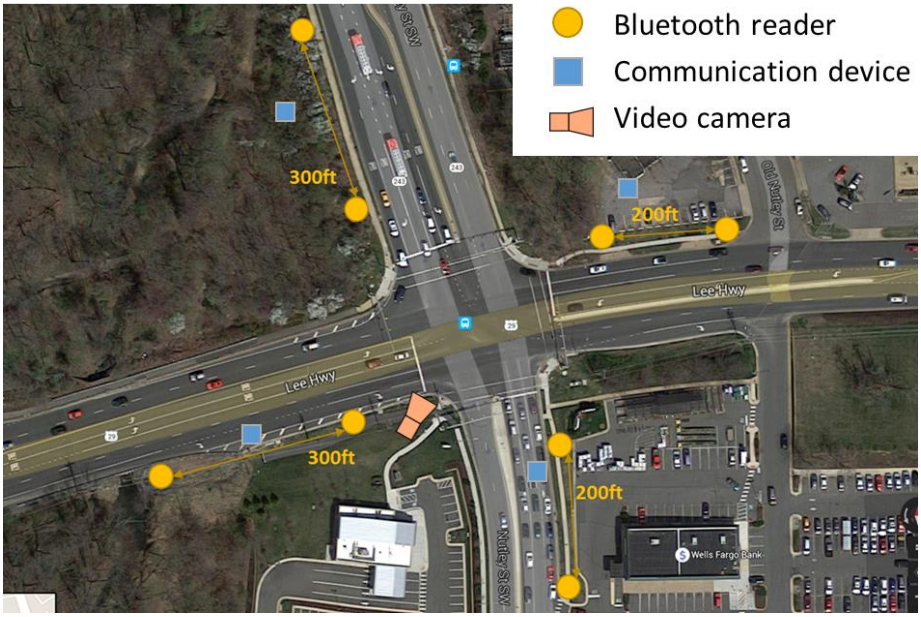


Figure 13. Sensor installation at the study area to investigate the MPR [33].

As shown in Table 5, we counted 762 and 1,784 vehicles traveling eastbound and westbound for an hour, respectively. Of these vehicles, 148 and 224 unique MAC addresses were collected for the same hour. Hence, the data collected at the study area revealed that the MPRs were 19.42% for eastbound and 12.56% for westbound. We found issues that the MAC addresses of iOS devices could be collected only when the Bluetooth pairing mode was turned on; however, if the device was in an enclosed space such as a bag or a pocket, the MAC address could not be collected. If the MAC addresses could have been collected from the iOS devices as well, we would have expected the field MPRs to exceed the minimum MPR. Given that market penetrations and the percentage of matched vehicles from the field testing were much lower than the minimum required MPR, it is recommended not to implementat the CTR algorithm in the field.

Table 5. Collected Unique MAC Addresses and Computed MPR of the Study Area

Direction	Volume (vph)	Number of unique MAC address	MPR (%)
Eastbound	762	148	**19.42**
Westbound	1784	224	**12.56**

Conclusion and Recommendations

To verify the feasibility of field implementation in the near future, the project team enhanced and evaluated a CTR real-time intersection control algorithm under various conditions, considering MPR, traffic demand, and types of available data. An existing intersection in the Northern Virginia Connected Vehicle Test Bed was simulated within VISSIM under the current traffic signal timing plans and volumes of peak and off-peak hours. Two CTT estimation techniques, SKF and AKF,

were applied for each phase in the CTR algorithm. In addition, a configuration based on HILS was proposed to test the feasibility of implementing the CTR algorithm in the field. Following are findings from this evaluation.

1. The CTR algorithm improved mobility compared with the actuated traffic signal control when MPR exceeded 30% with the SKF and 20% with the AKF. At 100% MPR, total travel time, average speed, and delay were significantly enhanced when compared with the current actuated traffic signal control. Without installation of infrastructure sensors, the CTR algorithm could be considered if the intersection has high traffic demand with 50%–60% MPR.

2. We found that the AKF outperformed the SKF at peak hour because it reduced the uncertainties with the process and observation noise statistics. Although environmental sustainability was not much improved, the CTR algorithm is highly expected to improve mobility performance under a CV environment.

3. As expected, the CTR algorithm's performance largely depends on the MPR because information from CVs is a key factor of the CTR algorithm. Given the low market penetrations and the percentage of matched vehicles found in the field testing, it is not currently recommended to consider implementation of the CTR algorithm in the field.

4. However, the team found that the perfect MPR requirement for the CTR algorithm could be relaxed (i) when data were collected from both CV and infrastructure sensors, and (ii) when AKF was adopted in the CTR algorithm.

5. The team could not implement the proposed algorithm because the measured field MPR was much lower than the minimum required MPR. Instead, the team developed an implementation plan for the CTR algorithm that can be easily adopted by traffic engineers once the field MPR reaches minimum requirements. Also, the system developed in this research is ready to be deployed and can be used for testing any new control algorithms within a risk-free research environment.

Although useful insights were found in this research, there are several challenges for successful implementation of the CTR algorithm in the field. The performance of communication devices should be considered because it affects the reliability of the data collected from CVs. For reliable information, advanced communication protocols, such as Dedicated Short-Range Communications (DSRC), might be needed to minimize packet losses and latencies of data delivery. The findings of this research are expected to be of great use in trying to implement the CTR algorithm with minor modifications in the field to improve network performances.

Implementation Plan for the CTR Algorithm

Although a full field test of was not feasible at the time of the research due to the low MPR of CVs, the research team developed an implementation plan that can be used in the future.

Figure 14 illustrates the implementation plan for the CTR algorithm. Once Bluetooth readers read MAC addresses, a Zigbee that is set up as a router and connected to a Raspberry Pi sends these MAC addresses to another Zigbee that is set up as a coordinator and connected to a server. As shown in the appendix, the "*BTcollect*" code sorts vehicles' MAC addresses from observed data by signal strength and saves them in a database. The "*TTCalc*" code matches unique MAC addresses upstream and downstream of the intersection for each approach and calculates the vehicle's travel time. The CTR algorithm uses these travel time data to determine the next green phase.

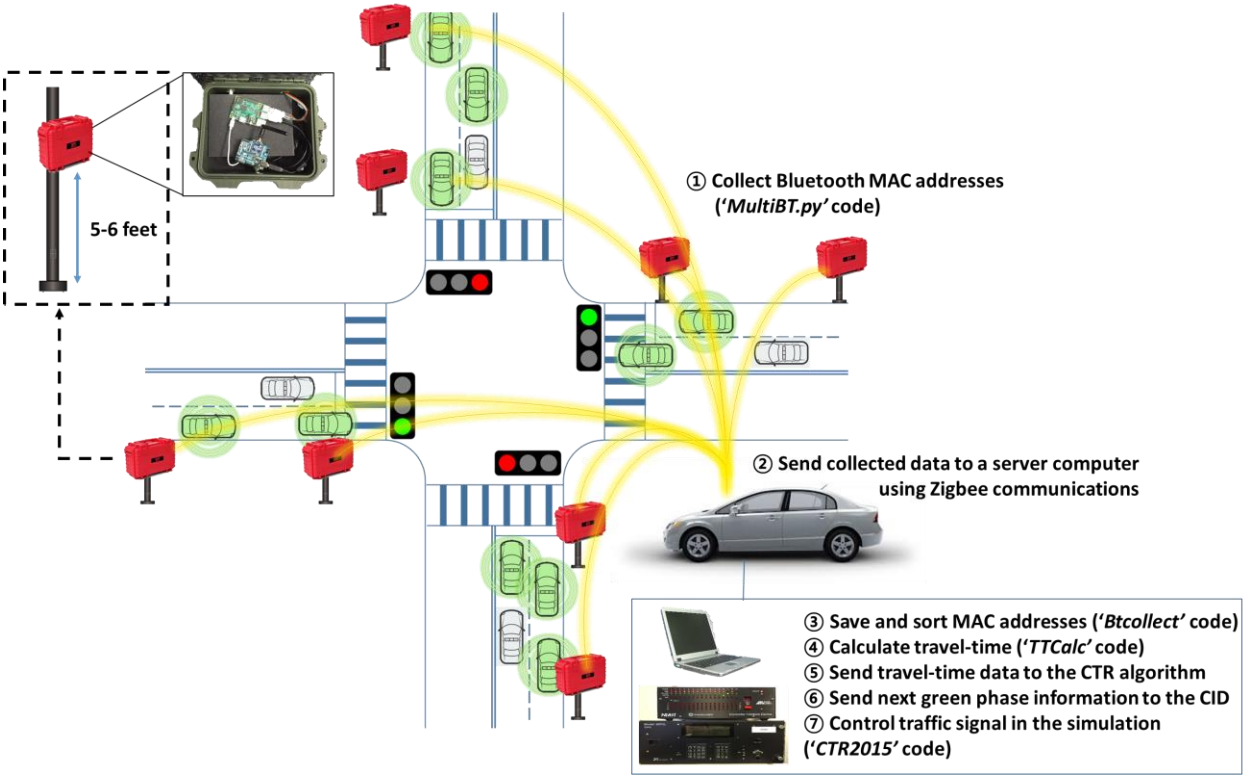


Figure 14. Implementation plan of the CTR algorithm at an intersection.

Components

To implement the CTR algorithm, the following components are required.

- Devices
  - Raspberry Pi and Micro SD card
  - Bluetooth reader
  - Communication devices: Zigbee shield, Zigbee module, and Arduino
  - Traffic signal controller hardware, CID

- Source codes
    - CTR algorithm code
    - Python code for operating sensors
    - Travel time calculation code

Reference

[1] Federal Highway Administration, "Urban Congestion Reports - Operations Performance Measurement - FHWA Operations."

[2] P. Mirchandani and L. Head, "A real-time traffic signal control system: architecture, algorithms, and analysis," *Transp. Res. Part C Emerg. Technol.*, vol. 9, no. 6, pp. 415–432, Dec. 2001.

[3] K. Fehon and J. Peters, "Adaptive Traffic Signals, Comparison and Case Studies," presented at the the ITE Western District Annual Meeting, San Francisco, California, 2010.

[4] J. W. C. van Lin, "Incremental and online learning through extended kalman filtering with constraint weights for freeway travel time prediction," in *IEEE Intelligent Transportation Systems Conference, 2006. ITSC '06*, 2006, pp. 1041–1046.

[5] C. Antoniou, M. Ben-Akiva, and H. Koutsopoulos, "Online Calibration of Traffic Prediction Models," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1934, pp. 235–245, Jan. 2005.

[6] Y. Wang and M. Papageorgiou, "Real-time freeway traffic state estimation based on extended Kalman filter: a general approach," *Transp. Res. Part B Methodol.*, vol. 39, no. 2, pp. 141–167, Feb. 2005.

[7] Chu L., Oh J., and Recker W., "Adaptive Kalman filter based freeway travel time estimation," presented at the the 84th TRB Annual Meeting, Washington, DC, 2005.

[8] J. Guo, W. Huang, and B. M. Williams, "Adaptive Kalman filter approach for stochastic short-term traffic flow rate prediction and uncertainty quantification," *Transp. Res. Part C Emerg. Technol.*, vol. 43, Part 1, pp. 50–64, Jun. 2014.

[9] R.-M. Hage, D. Betaille, F. Peyret, and D. Meizel, "Unscented Kalman filter for urban network travel time estimation," *Procedia - Soc. Behav. Sci.*, vol. 54, pp. 1047–1057, Oct. 2012.

[10] H. Chen and S. Grant-Muller, "Use of sequential learning for short-term traffic flow forecasting," *Transp. Res. Part C Emerg. Technol.*, vol. 9, no. 5, pp. 319–336, Oct. 2001.

[11] H. Yin, S. C. Wong, J. Xu, and C. K. Wong, "Urban traffic flow prediction using a fuzzy-neural approach," *Transp. Res. Part C Emerg. Technol.*, vol. 10, no. 2, pp. 85–98, Apr. 2002.

[12] Jiang X. and Adeli H., "Dynamic Wavelet Neural Network Model for Traffic Flow Forecasting," *J. Transp. Eng.*, vol. 131, no. 10, pp. 771–779, 2005.

[13] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, "Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach," *Transp. Res. Part C Emerg. Technol.*, vol. 13, no. 3, pp. 211–234, Jun. 2005.

[14] Dunne S. and Ghosh B., "Regime-Based Short-Term Multivariate Traffic Condition Forecasting Algorithm," *J. Transp. Eng.*, vol. 138, no. 4, pp. 455–466, 2012.

[15] L. Zhang, Q. Liu, W. Yang, N. Wei, and D. Dong, "An Improved K-nearest Neighbor Model for Short-term Traffic Flow Prediction," *Procedia - Soc. Behav. Sci.*, vol. 96, pp. 653–662, Nov. 2013.

[16] I. H. Zohdy and H. Rakha, "Game theory algorithm for intersection-based cooperative adaptive cruise control (CACC) systems," in *2012 15th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2012, pp. 1097–1102.

[17] J. Lee and B. Park, "Development and Evaluation of a Cooperative Vehicle Intersection Control Algorithm Under the Connected Vehicles Environment," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 81–90, Mar. 2012.

[18] S. Ilgin Guler, M. Menendez, and L. Meier, "Using connected vehicle technology to improve the efficiency of intersections," *Transp. Res. Part C Emerg. Technol.*, vol. 46, pp. 121–131, Sep. 2014.

[19] Y. Dujardin, D. Vanderpooten, and F. Boillot, "A multi-objective interactive system for adaptive traffic control," *Eur. J. Oper. Res.*, vol. 244, no. 2, pp. 601–610, Jul. 2015.

[20] Y. Feng, K. L. Head, S. Khoshmagham, and M. Zamanipour, "A real-time adaptive signal control in a connected vehicle environment," *Transp. Res. Part C Emerg. Technol.*, vol. 55, pp. 460–473, Jun. 2015.

[21] Planung Transport Verkehr (PTV), *VISSIM 5.40 User Manual*. Karlsruhe, Germany: PTV, 2009.

[22] Lee J., Park B., and Yun I., "Cumulative Travel-Time Responsive Real-Time Intersection Control Algorithm in the Connected Vehicle Environment," *J. Transp. Eng.*, vol. 139, no. 10, pp. 1020–1029, 2013.
[23] A. Stathopoulos and M. G. Karlaftis, "A multivariate state space approach for urban traffic flow modeling and prediction," *Transp. Res. Part C Emerg. Technol.*, vol. 11, no. 2, pp. 121–135, Apr. 2003.
[24] S. Shekhar and B. Williams, "Adaptive Seasonal Time Series Models for Forecasting Short-Term Traffic Flow," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2024, pp. 116–125, Jan. 2008.
[25] J. Guo and B. Williams, "Real-Time Short-Term Traffic Speed Level Forecasting and Uncertainty Quantification Using Layered Kalman Filters," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2175, pp. 28–37, Dec. 2010.
[26] O. Ziqiang, "Estimation of variance and covariance components," *Bull. Géod.*, vol. 63, no. 2, pp. 139–148, Jun. 1989.
[27] K. Xiong, C. L. Wei, and L. D. Liu, "Robust multiple model adaptive estimation for spacecraft autonomous navigation," *Aerosp. Sci. Technol.*, vol. 42, pp. 249–258, Apr. 2015.
[28] U. Bandara, M. Hasegawa, M. Inoue, H. Morikawa, and T. Aoyama, "Design and implementation of a Bluetooth signal strength based location sensing system," in *2004 IEEE Radio and Wireless Conference*, 2004, pp. 319–322.
[29] S. Zhou and J. K. Pollard, "Position measurement using Bluetooth," *IEEE Trans. Consum. Electron.*, vol. 52, no. 2, pp. 555–558, May 2006.
[30] D. Bullock, B. Johnson, R. B. Wells, M. Kyte, and Z. Li, "Hardware-in-the-loop simulation," *Transp. Res. Part C Emerg. Technol.*, vol. 12, no. 1, pp. 73–89, Feb. 2004.
[31] J. Lee, Z. Zhong, B. Du, S. Gutesa, and K. Kim, "Low-Cost and Energy-Saving Wireless Sensor Network for Real-Time Urban Mobility Monitoring System," *J. Sens.*, vol. 2015, p. e685786, Sep. 2015.
[32] "Connected vehicle Virginia test bed," *Connected Vehicle/Infrastructure University Transportation Center (CVI-UTC)*. [Online]. Available: http://cvi-utc.org/?q=node/36. [Accessed: 04-Apr-2016].
[33] "Google Maps," *Google Maps*. [Online]. Available: https://www.google.com/maps/@38.0400874,-78.4849739,13z. [Accessed: 04-Apr-2016].
[34] Ahn K., Rakha H., Trani A., and Van Aerde M., "Estimating Vehicle Fuel Consumption and Emissions based on Instantaneous Speed and Acceleration Levels," *J. Transp. Eng.*, vol. 128, no. 2, pp. 182–190, 2002.

Appendix

Source Code for Operating the CTR Algorithm

Setting a Bluetooth Reader

Python Code: Serialtest.py

```
import time
import serial
import sys

# configure the serial connections (the parameters differs on the device you are connecting to)
if len(sys.argv) !=3:
   print "Wrong Argument. Needs a port number and Baud rate"
   sys.exit()

if not "/dev" in sys.argv[1]:
   print "Require correct device name. e.g., /dev/ttyUSB1"
   sys.exit()

ser = serial.Serial(
   port=sys.argv[1],\
   baudrate=int(sys.argv[2]),\
   parity=serial.PARITY_NONE,
   stopbits=serial.STOPBITS_ONE,
   bytesize=serial.EIGHTBITS,\
   timeout=0)

if ser.isOpen():
   ser.close()

ser.open()
ser.isOpen()

print 'Enter your commands below.\r\nInsert "byebye" to leave the application.'

input=1
while 1 :
   # get keyboard input
   input = raw_input(">> ")
      # Python 3 users
      # input = input(">> ")
   if input == 'byebye':
      ser.close()
      exit()
   else:
      # send the character to the device
      ser.write(input + '\r\n')
      out = ''
```

```
    # let's wait one second before reading output (let's give device time to answer)
    time.sleep(1)
    while ser.inWaiting() > 0:
      out += ser.read(1)

    if out != '':
      print ">>" + out
```

Collecting Bluetooth MAC Addresses

Python Code: MultiBTs.py

```
import thread
import time
import serial
import sys
import os

# Bluetooth Module1
def BTStart(brate, port_num, fn, btid):
  BTSerial1 = serial.Serial(
    port=port_num,\
    baudrate=brate,\
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,\
    timeout=0)

  if BTSerial1.isOpen():
    BTSerial1.close()

  BTSerial1.open()
  BTSerial1.isOpen()

  # Initialize
  print ("Testing....."+BTSerial1.port)

  BTSerial1.write('AT+NAME\r\n')
  time.sleep(0.1)
  out = ''
  tmpmsg = ''
  while BTSerial1.inWaiting() > 0:
    out = BTSerial1.readline()
    if not "OK" in out:
      print out.replace('\n',"")

  BTSerial1.write('AT+ROLE\r\n')
  time.sleep(0.1)
  out=BTSerial1.readlines()
  for tmpmsg in out:
    if not "OK" in tmpmsg:
      print tmpmsg.replace('\n',"")
```

```
BTSerial1.write('AT+INQM=0,10,4\r\n')
time.sleep(0.1)
out=BTSerial1.readlines()
for tmpmsg in out:
   if not "OK" in tmpmsg:
      print tmpmsg.replace('\n',"")

BTSerial1.write('AT+STATE\r\n')
time.sleep(0.1)
while BTSerial1.inWaiting() > 0:
   out = BTSerial1.readline()
   if not "OK" in out:
      tmpmsg +=out
      print out.replace('\n',"")

if "INITIALIZED" in tmpmsg:
   BTSerial1.write('AT+INIT\r\n')
   time.sleep(0.1)
   out = BTSerial1.readline()
   print out.replace('\n',"") +". Initialization Done"

tmp=''
timestamp =''
#timestamp1 =''
#timestamp2 =''

while True:
   data_=''
   #timegap1 = time.clock()
   BTSerial1.write('AT+INQ\r\n')
   timestamp=time.ctime()
   time.sleep(4*1.28+0.1)
   out = BTSerial1.readlines()
   #timegap2 = time.clock()
   output = open(fn,"a")
   output.write(btid+"|"+BTSerial1.port+"|"+timestamp+"|")
   for tmpmsg in out:
      if not "OK" in tmpmsg:
         tmp = tmpmsg.replace('\n',"")
         tmp = tmp.replace('\r',"")
         tmp = tmp.replace("+INQ:","")
         data_+=tmp+"|"
   #timegap3 = time.clock()
   #gap21 = timegap2-timegap1
   #gap31 = timegap3-timegap1
   #gap32 = timegap3-timegap2
   #print '%f %f %f %s' %(gap21, gap31, gap32, BTSerial1.port+"|"+timestamp+"|"+data_+'\n')
   print btid+"|"+BTSerial1.port+"|"+timestamp+"|"+data_+'\n'
   output.write(data_+'\n')
   output.close();
   time.sleep(4*1.28)
```

```
# Main Run

if len(sys.argv) !=3:
   print "wrong argument setting"
   print "  python MultiBT.py Baud_Rate port#1(/dev/ttyUSB1) port#2(/dev/ttyUSB2)"
   sys.exit()

print " Start "
os.popen("echo ds1307 0x68 | sudo tee /sys/class/i2c-adapter/i2c-1/new_device")
os.popen("sudo hwclock --hctosys")

time.sleep(30)

strtmp_ = os.popen("sudo cat /etc/network/interfaces | grep address").read()
BTId = strtmp_.split("s ")[1]
BTId = BTId.replace('\n',"")
BTId = BTId.replace('\r',"")
#BTId = strtmp_.replace("inet ","")

output = open(BTId+".out","a")
output.write("\n\n")
output.write("# Begin Scanning at "+time.ctime()+"\n")
output.close()

try:
   thread.start_new_thread(BTStart,(sys.argv[1], sys.argv[2],BTId+".out",BTId,))
   time.sleep(4*1.28+0.5)
   thread.start_new_thread(BTStart,(sys.argv[1], sys.argv[3],BTId+".out",BTId,))
except:
   print "Error"
while 1:
   pass
```

Calculating Kalman Filter Matrix

Matlab Code: Adaptive Kalman Filter

```
function [x_hat_k P_k dx_k diag_Q_hat] =
RunAdaptiveKalman3(al1,al2,at1,bl1,bt1,l,h,q,d,Mn,r,rx,rho,P_k_1,x_hat_k_1,u_k_1,Zk)
% from "Adaptive Kalman Filtering for INS/GPS"
% A.H Mohamed and K.P. Schwarz (1999)
% al1, al2 : Coefficents for Left-Turn's A matrix
% Scalar calculatin

  A=[al1 0 0 0 0 al2 0 0; % 1
    0 at1 0 0 0 0 0 0; %2
    0 0 al1 0 0 0 0 al2; %3
    0 0 0 at1 0 0 0 0; %4
    0 al2 0 0 al1 0 0 0; %5
    0 0 0 0 0 at1 0 0; %6
    0 0 0 al2 0 0 al1 0; %7
    0 0 0 0 0 0 0 at1]; %8

  %    V(volume)       L(lane)
  B=[bl1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 bt1 0 0 0 0 0 0 0 l 0 0 0 0 0 0 0;
    0 0 bl1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 bt1 0 0 0 0 0 0 0 0 l 0 0 0 0;
    0 0 0 0 bl1 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 bt1 0 0 0 0 0 0 0 0 l 0 0;
    0 0 0 0 0 0 bl1 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 bt1 0 0 0 0 0 0 0 0 l];

  H=[h*rho(1) 0 0 0 0 0 0 0;
    0 h*rho(2) 0 0 0 0 0 0;
    0 0 h*rho(3) 0 0 0 0 0;
    0 0 0 h*rho(4) 0 0 0 0;
    0 0 0 0 h*rho(5) 0 0 0;
    0 0 0 0 0 h*rho(6) 0 0;
    0 0 0 0 0 0 h*rho(7) 0;
    0 0 0 0 0 0 0 h*rho(8)];

  Q=[q(1) 0 0 0 0 0 0 0;
    0 q(2) 0 0 0 0 0 0;
    0 0 q(3) 0 0 0 0 0;
    0 0 0 q(4) 0 0 0 0;
    0 0 0 0 q(5) 0 0 0;
    0 0 0 0 0 q(6) 0 0;
    0 0 0 0 0 0 q(7) 0;
    0 0 0 0 0 0 0 q(8)];

  rLen = length(r)/8; % Memory Size

  N=rLen;

  if (N>Mn)
```

```
    N=Mn;
  end

  c=1;
  rsum=0;
  dsum=0;
  for i=1:N,
    vk = zeros(8,1);
    dxk = zeros(8,1);
    for j=1:8
      vk(j) = r(c);
      dxk(j) = d(c);
      c=c+1;
    end
    rsum=rsum+vk'*vk;
    dsum=dsum+dxk'*dxk;
  end

  C_hat_vk = rsum/N;
  C_hat_dxk = dsum/N;

  if C_hat_vk == 0
    C_hat_vk = rx;
  end

  R=C_hat_vk*eye(8,8)+H*P_k_1*H';
  x_pri_hat_k = A*x_hat_k_1'+B*u_k_1';
  P_pri_k = A*P_k_1*A'+Q;
  K_k = P_pri_k*H'*inv(H*P_pri_k*H'+R);
  x_hat_k = x_pri_hat_k + K_k*(Zk'-H*x_pri_hat_k);
  P_k = (eye(8,8)-K_k*H)*P_pri_k;
  dx_k = x_hat_k- x_pri_hat_k;
  Q_hat = C_hat_dxk*eye(8,8)+P_k-A*P_k_1*A';
  diag_Q_hat = abs(diag(Q_hat));
end
```

Matlab Code: Standard Kalman Filter

```
function [x_hat_k P_k dx_k diagQ] =
RunKalmanNOGR(al1,al2,at1,bl1,bt1,l,h,qlt,qth,r,rho,P_k_1,x_hat_k_1,u_k_1,Zk)


   A=[al1 0 0 0 0 al2 0 0; % 1
     0 at1 0 0 0 0 0 0; %2
     0 0 al1 0 0 0 0 al2; %3
     0 0 0 at1 0 0 0 0; %4
     0 al2 0 0 al1 0 0 0; %5
     0 0 0 0 0 at1 0 0; %6
     0 0 0 al2 0 0 al1 0; %7
     0 0 0 0 0 0 0 at1]; %8

   %    V(volume)       L(lane)
   %------------------- ---------------
   B=[bl1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
```

```
    0 bt1 0 0 0 0 0 0 0 l 0 0 0 0 0 0;
    0 0 bl1 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 bt1 0 0 0 0 0 0 0 l 0 0 0 0;
    0 0 0 0 bl1 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 bt1 0 0 0 0 0 0 0 l 0 0;
    0 0 0 0 0 0 bl1 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 bt1 0 0 0 0 0 0 0 l];


  H=[h*rho(1) 0 0 0 0 0 0 0;
    0 h*rho(2) 0 0 0 0 0 0;
    0 0 h*rho(3) 0 0 0 0 0;
    0 0 0 h*rho(4) 0 0 0 0;
    0 0 0 0 h*rho(5) 0 0 0;
    0 0 0 0 0 h*rho(6) 0 0;
    0 0 0 0 0 0 h*rho(7) 0;
    0 0 0 0 0 0 0 h*rho(8)];

  Q=[qlt 0 0 0 0 0 0 0;
    0 qth 0 0 0 0 0 0;
    0 0 qlt 0 0 0 0 0;
    0 0 0 qth 0 0 0 0;
    0 0 0 0 qlt 0 0 0;
    0 0 0 0 0 qth 0 0;
    0 0 0 0 0 0 qlt 0;
    0 0 0 0 0 0 0 qth];

  R = r*eye(8,8);
  %P_k_1=eye(8,8);

  %x_k_1 = [10 11 16 13 14 15 16 17];
  %u_k_1 = [11 12 13 14 15 16 17 18 0 1 0 2 0 3 0 4];

  x_pri_hat_k = A*x_hat_k_1'+B*u_k_1';
  P_pri_k = A*P_k_1*A'+Q;
  K_k = P_pri_k*H'*inv(H*P_pri_k*H'+R);
  x_hat_k = x_pri_hat_k + K_k*(Zk'-H*x_pri_hat_k);
  P_k = (eye(8,8)-K_k*H)*P_pri_k;
  dx_k = x_hat_k - x_pri_hat_k;
  diagQ = abs(diag(Q));
end
```

Operating the CTR Algorithm

C# Code: CTR2015

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RunAdaptiveKalman;
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using VISSIM_COMSERVERLib;
using System.IO;

namespace CTR2015
{
    class Program
    {
        static public Vissim vissim;
        public static Dictionary<int, double> Z = new Dictionary<int, double>();
        public static Dictionary<int, double> u = new Dictionary<int, double>();

        public static double[] MatZ = new double[8];
        public static double[] MatU;

        public static int[] LinkIDs = { 1000002, 1000009, 1000005, 1000012, 1000008,
1000003, 1000011, 1000006 };
        public static StreamWriter sw;
        public static double[] x_hat_pos_k_1 = new double[8]; // x^(k-1)
        public static double[] P_k_1 = new double[8]; //P(k-1)
        public static double[,] MatP_k_1 = new double[8, 8]; //P(k-1) for matrix
        public static double[] P_K = new double[8]; // P(k)
        public static double[] K_k = new double[8]; //K(k)
        public static double[] x_hat_pri_k = new double[8]; // x^-(k)

        public static double A = 0.916;
        public static double Al = 0.965;
        public static double B = 1.07;
        public static double Bl = 1.93;

        public static double[] MatTTime = new double[8];
        public static double[] MatRho = new double[8];

        public static int[] GreenPhase = new int[8];


        public static int Mn = 30; // memory size
        public static double K;
        public static Dictionary<int, List<double>> r = new Dictionary<int,
List<double>>();
        public static Dictionary<int, List<double>> MatR = new Dictionary<int,
List<double>>();
        public static Dictionary<int, List<double>> Matdk = new Dictionary<int,
List<double>>();
        public static Dictionary<int, List<double>> q = new Dictionary<int,
List<double>>();
        public static List<double> ListR;
        public static List<double> Listdx_k;
        public static int m = 1799;
```

```
        public static double[,] _diagQ_ = new double[1, 8];

        public static int rCounter = 0;

        public static RunAdaptiveKalman.Class1 OutEstimated = new Class1();

        public static StreamWriter swlog = new StreamWriter(@"c:\feedback\akf\log.csv");

        static void Main()
        {
            int i, j;

            //double[] randomnum = new double[100];

            int[] NextLinkIDs = { 1, 4, 5, 8 };
            double[,] VolSce = new double[51, 16];
            //Dictionary<int, int> RouteMap = new Dictionary<int, int>();

            string fn = @"C:\feedback\network.inp";
            string fn_ini = @"C:\feedback\vissim.ini";
            string fn_volsce = @"C:\feedback\VolumeScenarios_SORTED.csv";
            int scnNum = 50;
            int repNum = 6;
            int seed = 105;
            bool ani = true;
            double mp = 1; // Market %

            string tic;

            string[] args = new string[5];
            args[0] = "20";
            args[1] = "6";
            args[2] = "105";
            args[3] = "1";
            args[4] = "70";

            //fn = args[0];
            //fn_ini = args[1];
            //fn_volsce = args[2];
            scnNum = Convert.ToInt32(args[0]);
            repNum = Convert.ToInt32(args[1]);
            seed = Convert.ToInt32(args[2]);

            if (args[3] == "0") ani = false;

            mp = Convert.ToDouble(args[4]) / 100.0;


            InitDictionaries();

            //double[] u = new double[nphase];

            VolSce = LoadVolScen(fn_volsce, VolSce);

            vissim = new Vissim();

            InitVissim(seed, ani, fn, fn_ini, VolSce, scnNum, mp);

            //tic = DateTime.Now.Ticks.ToString();
```

```
            sw = new StreamWriter(@"c:\feedback\AKF\AKFRun\akf_" + scnNum + "_" + repNum +
"_" + seed + "_" + args[4] + "p.csv");

            for (i = 1; i <= 3600; i++)
            {

                KF_EstimateTotalTravelTime_Matrix_woGreen();
                //AKF_EstimateTotalTravelTime_Matrix_woGreen();
                if (!SetSignalTime(5)) // prediction free w/ LT treatment
                    break;
            }

            vissim.Simulation.Stop();
            vissim.Exit();

            File.Copy(@"c:\feedback\network.npe", @"c:\feedback\AKF\AKFRUN\akf_" + scnNum +
"_" + repNum + "_" + seed + "_" + args[4] + "p.npe", true);
            sw.Close();

            Console.WriteLine("Done(akf_" + scnNum + "_" + repNum + "_" + args[4] + ")");

            swlog.Close();
        }

        private static void AKF_EstimateTotalTravelTime_Matrix_woGreen()
        {
            // Matrix Setting
            // --------- Covariances
            double Q = 2660.0;
            double Ql = 376.9;
            double R = 207.96;

            // -------- H matrix
            double H = 0.999;

            // -------- A matrix
            double al1 = 0.982;
            double al2 = -0.00457;
            double at1 = 0.916;

            // --------- B matrix
            double bl1 = 1.96; // for LT volumes
            double bt1 = 1.85; // for TH volume
            double nl = -6.44; // for TH Num Lanes (all 2)


            object VehIDs;
            int j;
            double vtime = 0.0;
            double ttime = 0.0;
            int vid;
            double N = 0.0;
            double n = 0.0;
            double rho = 0.0;

            MWArray[] OptOut = null;
            MWNumericArray x_hat_k = new MWNumericArray();
            MWNumericArray P_k = new MWNumericArray();
            MWNumericArray dx_k = new MWNumericArray();
            MWNumericArray diagQ = new MWNumericArray();
```

```
            // VehIDs = vissim.Net.Vehicles.get_IDs("LINK", lk);


            int i = 0;

            double TrueTT = 0.0;

            MatTTime = new double[8];
            MatRho = new double[8];
            MatU = new double[16];
            MatZ = new double[8];



            for (int l = 8; l < 16; l = l + 2)
                MatU[l + 1] = 2; // Number of lanes


            foreach (int lk in LinkIDs)
            {
                // to obtain true travel time
                VehIDs = vissim.Net.Vehicles.get_IDs("LINk", lk);


                MatU[i] = Convert.ToDouble(((object[])(VehIDs)).Length);


                for (int idx = 0; idx < MatU[i]; idx++)
                {
                    vid = Convert.ToUInt16(((object[])(VehIDs))[idx]);
                    MatTTime[i] +=
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("ELAPSEDTIME")) / 3.6; ;

                }

                // to obtain equipped cars' travel time

                VehIDs = vissim.Net.Links.GetLinkByNumber(lk).GetVehicles().get_IDs("TYPE",
1001);
                n = Convert.ToDouble(((object[])(VehIDs)).Length);


                if (MatU[i] > 0)
                    MatRho[i] = n / MatU[i];
                else
                    MatRho[i] = 0.0;


                for (j = 0; j < Convert.ToInt16(((object[])(VehIDs)).Length); j++)
                {
                    vid = Convert.ToUInt16(((object[])(VehIDs))[j]);

                    vtime =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("ELAPSEDTIME")) /
3.6;//mps
                    //vmile =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("TOTALDISTANCE")) /
3.6;//mps
                    MatZ[i] += vtime;

                }
```

```
            i++;
        }

        if (rCounter > Mn)
        {

            ListR = new List<double>();
            Listdx_k = new List<double>();

            foreach (int idx in MatR.Keys.Reverse<int>())
            {
                for (int idz = 0; idz < 8; idz++)
                {
                    ListR.Add(MatR[idx][idz]);

                }
            }

            foreach (int idx in Matdk.Keys.Reverse<int>())
            {
                for (int idz = 0; idz < 8; idz++)
                {
                    Listdx_k.Add(Matdk[idx][idz]);

                }
            }



            double[] ArrayR;
            double[] Arraydx_k;
            double[] ArraydiagQ = new double[8];

            //swlog.Write("qx=[");
            //for (int g = 0; g < 8; g++)
            //{
            //    ArraydiagQ[g] = _diagQ_[g, 0];
            //    swlog.Write(" "+Math.Round(ArraydiagQ[g],1));
            //}
            //swlog.WriteLine("];");



            if (rCounter > Mn)
            {
                ArrayR = new double[Mn * 8];
                Arraydx_k = new double[Mn * 8];
                for (int c = 0; c < Mn * 8; c++)
                {
                    ArrayR[c] = ListR[c];
                    Arraydx_k[c] = Listdx_k[c];
                }

            }
            else
            {
                ArrayR = new double[rCounter * 8];
                Arraydx_k = new double[rCounter * 8];
                for (int c = 0; c < rCounter * 8; c++)
                {
```

```
                        ArrayR[c] = ListR[c];
                        Arraydx_k[c] = Listdx_k[c];
                    }
                }

                OptOut = OutEstimated.RunAdaptiveKalman3(4, al1, al2, at1, bl1, bt1, nl, H,
(MWNumericArray)ArraydiagQ,
                                                         (MWNumericArray)Arraydx_k, Mn,
(MWNumericArray)ArrayR, R, (MWNumericArray)MatRho, (MWNumericArray)MatP_k_1,
(MWNumericArray)x_hat_pos_k_1,
                                                         (MWNumericArray)MatU, (MWNumericArray)MatZ);
            }
            else
                OptOut = OutEstimated.RunKalmanNOGR(4, al1, al2, at1, bl1, bt1, nl, H, Ql,
Q, R,
                                    (MWNumericArray)MatRho, (MWNumericArray)MatP_k_1,
                                    (MWNumericArray)x_hat_pos_k_1, (MWNumericArray)MatU,
(MWNumericArray)MatZ);



            x_hat_k = (MWNumericArray)OptOut[0];
            P_k = (MWNumericArray)OptOut[1];
            dx_k = (MWNumericArray)OptOut[2];
            diagQ = (MWNumericArray)OptOut[3];

            double[,] _x_hat_k_ = new double[1, 8];
            double[,] _MatP_k_ = new double[8, 8];
            double[,] _dx_k_ = new double[1, 8];


            _x_hat_k_ = (double[,])x_hat_k.ToArray(MWArrayComponent.Real);
            _MatP_k_ = (double[,])P_k.ToArray(MWArrayComponent.Real);
            _dx_k_ = (double[,])dx_k.ToArray(MWArrayComponent.Real);
            _diagQ_ = (double[,])diagQ.ToArray(MWArrayComponent.Real);

            MatR.Add(rCounter, new List<double>());
            Matdk.Add(rCounter, new List<double>());


            for (int o = 0; o < 8; o++)
            {
                MatR[rCounter].Add(MatZ[o] - H * MatRho[o] * x_hat_pos_k_1[o]);
                Matdk[rCounter].Add(_dx_k_[o, 0]);
            }

            rCounter++;




            for (int iter = 0; iter < 8; iter++)
            {
                sw.Write(MatTTime[iter] + "," + _x_hat_k_[iter, 0] + "," + MatZ[iter] + ","
+ MatRho[iter] + ",");

                if (_x_hat_k_[iter, 0] >= 0.0)
                    x_hat_pos_k_1[iter] = _x_hat_k_[iter, 0];
                else
                    x_hat_pos_k_1[iter] = 0.0;
```

```
            for (int g = 0; g < 8; g++)
                MatP_k_1[iter, g] = _MatP_k_[iter, g];
        }


        sw.WriteLine(vissim.Simulation.get_AttValue("ELAPSEDTIME"));
        sw.Flush();
    }

    private static void KF_EstimateTotalTravelTime_Matrix_woGreen()
    {
        // Matrix Setting
        // --------- Covariances
        double Q = 2660.0;
        double Q1 = 376.9;
        double R = 207.96;

        // -------- H matrix
        double H = 0.999;

        // -------- A matrix
        double a11 = 0.982;
        double a12 = -0.00457;
        double at1 = 0.916;

        // --------- B matrix
        double b11 = 1.96; // for LT volumes
        double bt1 = 1.85; // for TH volume
        double n1 = -6.44; // for TH Num Lanes (all 2)


        object VehIDs;
        int j;
        double vtime = 0.0;
        double ttime = 0.0;
        int vid;
        double N = 0.0;
        double n = 0.0;
        double rho = 0.0;

        MWArray[] OptOut = null;
        MWNumericArray x_hat_k = new MWNumericArray();
        MWNumericArray P_k = new MWNumericArray();

        // VehIDs = vissim.Net.Vehicles.get_IDs("LINK", lk);


        int i = 0;

        double TrueTT = 0.0;

        MatTTime = new double[8];
        MatRho = new double[8];
        MatU = new double[16];
        MatZ = new double[8];

        for (int l = 8; l < 16; l = l + 2)
            MatU[l + 1] = 2; // Number of lanes
```

```
            foreach (int lk in LinkIDs)
            {
                // to obtain true travel time
                VehIDs = vissim.Net.Vehicles.get_IDs("LINk", lk);


                MatU[i] = Convert.ToDouble(((object[])(VehIDs)).Length);


                for (int idx = 0; idx < MatU[i]; idx++)
                {
                    vid = Convert.ToUInt16(((object[])(VehIDs))[idx]);
                    MatTTime[i] +=
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("ELAPSEDTIME")) / 3.6; ;

                }

                // to obtain equipped cars' travel time

                VehIDs = vissim.Net.Links.GetLinkByNumber(lk).GetVehicles().get_IDs("TYPE",
1001);
                n = Convert.ToDouble(((object[])(VehIDs)).Length);


                if (MatU[i] > 0)
                    MatRho[i] = n / MatU[i];
                else
                    MatRho[i] = 0.0;


                for (j = 0; j < Convert.ToInt16(((object[])(VehIDs)).Length); j++)
                {
                    vid = Convert.ToUInt16(((object[])(VehIDs))[j]);

                    vtime =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("ELAPSEDTIME")) /
3.6;//mps
                    //vmile =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("TOTALDISTANCE")) /
3.6;//mps
                    MatZ[i] += vtime;

                }

                i++;
            }

            OptOut = OutEstimated.RunKalmanNOGR(2, al1, al2, at1, bl1, bt1, nl, H, Ql, Q,
R,
                                                (MWNumericArray)MatRho,
(MWNumericArray)MatP_k_1,
                                                (MWNumericArray)x_hat_pos_k_1,
(MWNumericArray)MatU, (MWNumericArray)MatZ);

            x_hat_k = (MWNumericArray)OptOut[0];
            P_k = (MWNumericArray)OptOut[1];

            double[,] _x_hat_k_ = new double[1, 8];
            double[,] _MatP_k_ = new double[8, 8];

            _x_hat_k_ = (double[,])x_hat_k.ToArray(MWArrayComponent.Real);
```

```
            _MatP_k_ = (double[,])P_k.ToArray(MWArrayComponent.Real);


            for (int iter = 0; iter < 8; iter++)
            {
                //sw.Write(MatTTime[iter] + "," + _x_hat_k_[iter, 0] + "," + MatZ[iter] +
"," + MatRho[iter] + ",");
                x_hat_pos_k_1[iter] = _x_hat_k_[iter, 0];
                for (int g = 0; g < 8; g++)
                    MatP_k_1[iter, g] = _MatP_k_[iter, g];
            }


            sw.WriteLine(K);
            sw.Flush();

        }
        private static void InitDictionaries()
        {
            for (int i = 0; i < 8; i++)
            {
                Z.Add(i, 0);
                u.Add(i, 0);
            }
            q.Add(0, new List<double>());
            r.Add(0, new List<double>());
        }

        private static bool SetSignalTime(int cl)
        {
            int nsg;
            int ns = 1;
            int i, j, k, l;
            int[] OldState;
            int[] NewState;
            int indicator1 = 0;
            int indicator2 = 0;

            int[,] P = { { 1, 5 }, { 1, 6 }, { 2, 5 }, { 2, 6 }, { 3, 7 }, { 3, 8 }, { 4, 7
}, { 4, 8 } };
            double[,] T;
            int a, b;

            double max = 0.0;

            GreenPhase = new int[8];

            nsg = Z.Count;

            OldState = new int[nsg];
            NewState = new int[nsg];
            T = new double[nsg, 3];

            for (i = 1; i <= nsg; i++)
            {
                if (GetSignalState(vissim, 1, i) == "Green")
                    OldState[i - 1] = 2;
                else
                    OldState[i - 1] = 3;
            }
```

```
            double a_ = 0.0;
            double b_ = 0.0;

            for (i = 0; i < nsg; i++)
            {
                a = P[i, 0] - 1;
                b = P[i, 1] - 1;

                if (x_hat_pos_k_1[a] >= 0.0)
                    a_ = x_hat_pos_k_1[a];
                else
                    a_ = 0.0;

                if (x_hat_pos_k_1[b] >= 0.0)
                    b_ = x_hat_pos_k_1[b];
                else
                    b_ = 0.0;


                T[i, 0] = a_ + b_;
                T[i, 1] = a;
                T[i, 2] = b;

            }


            for (j = 0; j < nsg; j++)
            {
                if (T[j, 0] > max)
                {
                    max = T[j, 0];
                    indicator1 = Convert.ToInt16(T[j, 1]); // indicate a phase to be GREEN
                    indicator2 = Convert.ToInt16(T[j, 2]); // indicate a phase to be GREEN
                }
            }

            GreenPhase[indicator1] = 1;
            GreenPhase[indicator2] = 1;


            if (!((indicator1 + 1) == 2 && (indicator2 + 1) == 6 || (indicator1 + 1) == 4
&& (indicator2 + 1) == 8))
            {
                cl = GetOptimalCl(indicator1 + 1, indicator2 + 1, vissim, cl, LinkIDs);
            }


            for (k = 0; k < nsg; k++)
            {
                if (indicator1 == k || indicator2 == k)
                    NewState[k] = 2;
                else
                    NewState[k] = 3;
            }

            indicator1 = 0;

            for (l = 0; l < nsg; l++)
            {
                if (OldState[l] != NewState[l])
                    indicator1 = l;
```

```
            }

            if (indicator1 > 0)
            {

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).set_AttValue("CYCLETIME", 0);

                for (k = 0; k < nsg; k++)
                {
                    if (OldState[k] == 2 && NewState[k] == 3) // green -> red : set yellow
                    {

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(k + 1).set_AttValue("TYPE", 1);

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(k + 1).set_AttValue("AMBER", 3);
                    }
                    else
                    {

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(k + 1).set_AttValue("TYPE", OldState[k]);
                    }
                }

                for (j = 1; j <= 3; j++)
                {
                    if (Convert.ToDouble(vissim.Simulation.get_AttValue("ELAPSEDTIME")) >=
m) return false;
                    vissim.Simulation.RunSingleStep();

                }


                for (k = 1; k <= nsg; k++)
                {

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(k).set_AttValue("TYPE", NewState[k - 1]);

                }

                for (j = 1; j <= (cl - 3); j++)
                {
                    if (Convert.ToDouble(vissim.Simulation.get_AttValue("ELAPSEDTIME")) >=
m) return false;
                    vissim.Simulation.RunSingleStep();

                }

            }
            else
            {
                for (k = 1; k <= nsg; k++)
                {

vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(k).set_AttValue("TYPE", NewState[k - 1]);

                }
```

```
                for (j = 1; j <= cl; j++)
                {
                    if (Convert.ToDouble(vissim.Simulation.get_AttValue("ELAPSEDTIME")) >=
m) return false;
                    vissim.Simulation.RunSingleStep();

                }

            }


            return true;

        }

        private static int GetOptimalCl(int indicator1, int indicator2, Vissim vissim, int
ocl, int[] LinkIDs)
        {
            object VehIDs;

            int cnt1, cnt2, cnt;
            int cl;
            int link1, link2;

            link1 = LinkIDs[indicator1 - 1];
            link2 = LinkIDs[indicator2 - 1];

            cnt1 = 0;
            cnt2 = 0;
            cnt = 0;

            if (indicator1 != 2 && indicator1 != 6 && indicator1 != 4 && indicator1 != 8)
            {
                VehIDs = vissim.Net.Vehicles.get_IDs("LINK", link1);
                cnt1 = Convert.ToInt16(((object[])(VehIDs)).Length);
            }

            if (indicator2 != 2 && indicator2 != 6 && indicator2 != 4 && indicator2 != 8)
            {
                VehIDs = vissim.Net.Vehicles.get_IDs("LINK", link2);
                cnt2 = Convert.ToInt16(((object[])(VehIDs)).Length);
            }

            if (cnt1 >= cnt2)
                cnt = cnt1;
            else
                cnt = cnt2;

            if (cnt < 3)
                cl = ocl;
            else
                cl = ocl + Convert.ToInt16(Convert.ToDouble(cnt) * 1.2);

            return Convert.ToInt16(cl);
        }

        private static double CalcTotalTravelTimeV2(Vissim vissim, int lk)
        {
            object VehIDs;
            int j;
```

```
            double vtime = 0.0;
            double ttime = 0.0;
            int vid;
            double llength;
            double vloc;

            VehIDs = vissim.Net.Vehicles.get_IDs("LINK", lk);

            llength = 3.28084 *
Convert.ToDouble(vissim.Net.Links.GetLinkByNumber(lk).get_AttValue("LENGTH"));

            for (j = 0; j < Convert.ToInt16(((object[])(VehIDs)).Length); j++)
            {
                vid = Convert.ToUInt16(((object[])(VehIDs))[j]);
                vloc = llength -
Convert.ToDouble(vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("LINKCOORD"));


                if (vloc < 200)
                {
                    vtime =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("ELAPSEDTIME")) /
3.6;//mps
                    //vmile =
((double)vissim.Net.Vehicles.GetVehicleByNumber(vid).get_AttValue("TOTALDISTANCE")) /
3.6;//mps

                    ttime = ttime + vtime;
                }
                //tmile = tmile + vmile;

            }

            //ttime = ttime/Convert.ToInt16(((object[])(VehIDs)).Length);
            //avgspd = tmile / ttime;

            return ttime;

        }

        private static double[,] LoadVolScen(string fn_volsce, double[,] A)
        {
            StreamReader sr = new StreamReader(fn_volsce);

            int i, j;
            string tmp = null;
            string[] line;

            for (i = 0; i < 51; i++)
            {
                tmp = sr.ReadLine();
                line = tmp.Split(",".ToCharArray());

                for (j = 0; j < 16; j++)
                {
                    A[i, j] = Convert.ToDouble(line[j]);
                }

            }

            sr.Close();
```

```
            return A;

        }


        private static void KillVissim(Vissim vissim)
        {
            vissim.Simulation.Stop();
            vissim.Exit();
        }


        private static string GetSignalState(Vissim vissim, int c, int g)
        {
            string s;
            s =
Convert.ToString(vissim.Net.SignalControllers.GetSignalControllerByNumber(c).SignalGroups.G
etSignalGroupByNumber(g).State);

            return s;

        }

        private static void InitVissim(int seed, bool visual, string fn, string fn_ini,
double[,] VolSce, int sce, double q_)
        {
            int i, j, k;
            int stype;


            vissim.LoadNet(fn, 0);
            vissim.LoadLayout(fn_ini);
            vissim.Simulation.Resolution = 1;
            vissim.Simulation.Period = 9999;
            //vissim.Simulation.BreakAt = 300;

            vissim.Evaluation.set_AttValue("NETPERFORMANCE", true);
            vissim.Evaluation.set_AttValue("DATACOLLECTION", false);
            //vissim.Evaluation.set_AttValue("LINK", true);
            //vissim.Evaluation.set_AttValue("DELAY", true);
            //vissim.Evaluation.set_AttValue("QUEUECOUNTER", true);

            vissim.Simulation.RandomSeed = seed;


            vissim.Graphics.set_AttValue("VISUALIZATION", visual);

            j = 0;

            for (i = 2; i <= 8; i = i + 2)
            {
                int l = 3;
                int vol;
                int tvol = 0;

                for (k = j; k < j + 3; k++)
                {
                    vol = Convert.ToInt16(VolSce[sce, k]);
```

```
vissim.Net.RoutingDecisions.GetRoutingDecisionByNumber(i).Routes.GetRouteByNumber(l).set_At
tValue1("RELATIVEFLOW", 1, vol);
                    tvol = tvol + vol;
                    l--;
                }
                vissim.Net.VehicleInputs.GetVehicleInputByNumber(i /
2).set_AttValue("VOLUME", tvol);


vissim.Net.TrafficCompositions.GetTrafficCompositionByNumber(i).set_AttValue1("RELATIVEFLOW
", 1001, q_); // % of equipped Car

vissim.Net.TrafficCompositions.GetTrafficCompositionByNumber(i).set_AttValue1("RELATIVEFLOW
", 1002, 1 - q_); // % of dumb car


                j = j + 4;
            }


//vissim.Net.SignalControllers.GetSignalControllerByNumber(1).set_AttValue("CYCLETIME",
cl);

            int cnt = r.Count;

            MatR.Add(0, new List<double>());
            Matdk.Add(0, new List<double>());
            rCounter++;


            vissim.Simulation.RunSingleStep(); // Added by JL on Apr 6 2015. To consider
VISSIM 5.4 or later. The simulation must be started to access the controller data.
            for (i = 1; i <= 8; i++)
            {
                if (i == 2 || i == 6)
                {

                    stype = 2;
                }
                else
                {
                    stype = 3;
                }


vissim.Net.SignalControllers.GetSignalControllerByNumber(1).SignalGroups.GetSignalGroupByNu
mber(i).set_AttValue("TYPE", stype);

                //Z[i - 1] = 0;
                //u[i - 1] = 0;
                x_hat_pos_k_1[i - 1] = 1.0;
                P_k_1[i - 1] = 1.0;

                for (int kk = 0; kk < 8; kk++)
                    MatP_k_1[i - 1, kk] = 1.0;


                x_hat_pri_k[i - 1] = 0.0;
                P_K[i - 1] = 0.0;
                K_k[i - 1] = 0.0;
```

```
                //r[cnt].Add(0.0);
                //q[cnt].Add(0.0);


                MatR[0].Add(0);
                Matdk[0].Add(0);
            }


            // -------- sort of warming up!
            for (i = 1; i < 20; i++)
            {
                vissim.Simulation.RunSingleStep();
            }

            GreenPhase[1] = 1;
            GreenPhase[5] = 1;

        }

    }
}
```

Collecting Vehicles' MAC Addresses

C# Code: BTcollect

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.IO;
using System.Net;
using System.Management;
using System.Data.OleDb;
using System.Data;

namespace _300ftbt
{
    class Program
    {
        public static Dictionary<int, string> MonthName = new Dictionary<int,
string>();
        static void Main(string[] args)
        {
            OleDbConnection connection = new OleDbConnection(); ;
            OleDbCommand command = new OleDbCommand();
            DataSet dataset = new DataSet();
            DateTime TimeNow = new DateTime();



            SerialPort XBeeConnection = new SerialPort();
            XBeeConnection.BaudRate = 9600;
```

```
            XBeeConnection.PortName = "COM17";
            XBeeConnection.Open();

            string[] token = null;
            string tstmp = null;
            string sender = null;
            string macad = null;

            connection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:/Projects/BTDataManager/DB/BTDB1.accdb;";
            connection.Open();

            SetMnth();
            string line = null;
            if (XBeeConnection.IsOpen)
            {
                while (true)
                {
                    line = XBeeConnection.ReadLine();

                    token = line.Split("|".ToCharArray());

                    if (token.Length > 2)
                    {
                        sender = token[0];
                        tstmp = token[1];

                        if (tstmp.Length != 24)
                            continue;
                        TimeNow = ConvertTime(tstmp);

                        try
                        {
                            for (int i = 2; i<token.Length - 1; i++)
                            {
                                string str = "INSERT INTO BTMacs(MAC,Sender,TStamp)"
+
                                    "VALUES( ('" + token[i] + "'),('" + sender +
"'),('" + TimeNow + "'))";
                                OleDbCommand insertCmd = new OleDbCommand(str,
connection);
                                insertCmd.ExecuteNonQuery();

                                Console.WriteLine(tstmp + "," + sender + "," +
token[i]);
                            }
                        }
                        catch (OleDbException)
                        {
                            Console.WriteLine("Error while uploading the data");
                        }
                    }

                }
```

```
            }

            connection.Close();

        }


        private static DateTime ConvertTime(string date_)
        {
            DateTime TimeNow = new DateTime();

            if (date_.Contains("  "))
                date_ = date_.Replace("  ", " ");

            int yy_;
            int mn_;
            int dd_;
            int hh_;
            int mm_;
            int ss_;
            string[] tstamp = null;


            mn_ = GetMnth(date_.Split(" ".ToCharArray())[1]);
            //if (mn_ == 0)
            //    return null;
            dd_ = int.Parse(date_.Split(" ".ToCharArray())[2]);

            tstamp = date_.Split(" ".ToCharArray())[3].Split(":".ToArray());

            hh_ = int.Parse(tstamp[0]);
            mm_ = int.Parse(tstamp[1]);
            ss_ = int.Parse(tstamp[2]);

            yy_ = int.Parse(date_.Split(" ".ToCharArray())[4]);

            TimeNow = new DateTime(yy_, mn_, dd_, hh_, mm_, ss_);

            return TimeNow;


        }

        public static void SetMnth()
        {
            MonthName.Add(1, "Jan");
            MonthName.Add(2, "Feb");
            MonthName.Add(3, "Mar");
            MonthName.Add(4, "Apr");
            MonthName.Add(5, "May");
            MonthName.Add(6, "Jun");
            MonthName.Add(7, "Jul");
            MonthName.Add(8, "Aug");
            MonthName.Add(9, "Sep");
            MonthName.Add(10, "Oct");
```

```
            MonthName.Add(11, "Nov");
            MonthName.Add(12, "Dec");
        }

        public static int GetMnth(string p)
        {
            int rt = 0;

            foreach (int i in MonthName.Keys)
            {
                if (MonthName[i] == p)
                    rt = i;

            }

            return rt;

        }

    }
}
```

C# Code: BTcollect

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.IO;
using System.Net;
using System.Management;
using System.Data.OleDb;
using System.Data;

namespace _300ftbt
{
    class Program
    {
        public static Dictionary<int, string> MonthName = new Dictionary<int,
string>();
        static void Main(string[] args)
        {
            OleDbConnection connection = new OleDbConnection(); ;
            OleDbCommand command = new OleDbCommand();
            DataSet dataset = new DataSet();
            DateTime TimeNow = new DateTime();



            SerialPort XBeeConnection = new SerialPort();
            XBeeConnection.BaudRate = 9600;
```

```
            XBeeConnection.PortName = "COM17";
            XBeeConnection.Open();

            string[] token = null;
            string tstmp = null;
            string sender = null;
            string macad = null;

            connection.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:/Projects/BTDataManager/DB/BTDB1.accdb;";
            connection.Open();

            SetMnth();
            string line = null;
            if (XBeeConnection.IsOpen)
            {
                while (true)
                {
                    line = XBeeConnection.ReadLine();

                    token = line.Split("|".ToCharArray());

                    if (token.Length > 2)
                    {
                        sender = token[0];
                        tstmp = token[1];

                        if (tstmp.Length != 24)
                            continue;
                        TimeNow = ConvertTime(tstmp);

                        try
                        {
                            for (int i = 2; i<token.Length - 1; i++)
                            {
                                string str = "INSERT INTO BTMacs(MAC,Sender,TStamp)"
+
                                    "VALUES( ('" + token[i] + "'),('" + sender +
"'),('" + TimeNow + "'))";
                                OleDbCommand insertCmd = new OleDbCommand(str,
connection);
                                insertCmd.ExecuteNonQuery();

                                Console.WriteLine(tstmp + "," + sender + "," +
token[i]);
                            }
                        }
                        catch (OleDbException)
                        {
                            Console.WriteLine("Error while uploading the data");
                        }
                    }

                }
```

```
            }

            connection.Close();

        }


        private static DateTime ConvertTime(string date_)
        {
            DateTime TimeNow = new DateTime();

            if (date_.Contains("  "))
                date_ = date_.Replace("  ", " ");

            int yy_;
            int mn_;
            int dd_;
            int hh_;
            int mm_;
            int ss_;
            string[] tstamp = null;


            mn_ = GetMnth(date_.Split(" ".ToCharArray())[1]);
            //if (mn_ == 0)
            //    return null;
            dd_ = int.Parse(date_.Split(" ".ToCharArray())[2]);

            tstamp = date_.Split(" ".ToCharArray())[3].Split(":".ToArray());

            hh_ = int.Parse(tstamp[0]);
            mm_ = int.Parse(tstamp[1]);
            ss_ = int.Parse(tstamp[2]);

            yy_ = int.Parse(date_.Split(" ".ToCharArray())[4]);

            TimeNow = new DateTime(yy_, mn_, dd_, hh_, mm_, ss_);

            return TimeNow;


        }

        public static void SetMnth()
        {
            MonthName.Add(1, "Jan");
            MonthName.Add(2, "Feb");
            MonthName.Add(3, "Mar");
            MonthName.Add(4, "Apr");
            MonthName.Add(5, "May");
            MonthName.Add(6, "Jun");
            MonthName.Add(7, "Jul");
            MonthName.Add(8, "Aug");
            MonthName.Add(9, "Sep");
            MonthName.Add(10, "Oct");
```

```
            MonthName.Add(11, "Nov");
            MonthName.Add(12, "Dec");
        }

        public static int GetMnth(string p)
        {
            int rt = 0;

            foreach (int i in MonthName.Keys)
            {
                if (MonthName[i] == p)
                    rt = i;

            }

            return rt;

        }

    }
}
```

Calculating Travel Time

C# Code: TTCalc

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.OleDb;
using System.Data;
using System.IO;

namespace TTCalc
{
    class Program
    {
        static void Main(string[] args)
        {
            OleDbConnection connection = new OleDbConnection(); ;
            OleDbCommand command = new OleDbCommand();
            OleDbDataAdapter adapter;
            DataSet dataset = new DataSet();

            DateTime TimeNow;

            Dictionary<string, Dictionary<string,string>> BTData = new
Dictionary<string, Dictionary<string, string>>(); // Dev,Mac,Timestamp

            string mnth = null;
            string date_ = null;
            string year = null;
            string hour_ = null;
```

```
            string mn_ = null;
            string ampm= null;

            string[] token = null;
            string tstmp = null;
            string up = @"""192.168.137.51""";
            string down = @"""192.168.137.52""";
            string macad = null;
            string str = null;
            string st = null;
            string ed = null;

            int upts = 0;
            int dnts = 0;
            int N = 1; // update interval in minute

            StreamWriter sw = new
StreamWriter(@"C:\Projects\BTDataManager\DB\out.csv");

            while (true)
            {
                connection.ConnectionString =
@"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:/Projects/BTDataManager/DB/BTDB1.accdb;";

                connection.Open();

                BTData = new Dictionary<string, Dictionary<string, string>>(); //
Dev,Mac,Timestamp

                BTData.Add(up, new Dictionary<string, string>());
                BTData.Add(down, new Dictionary<string, string>());

                Console.WriteLine("Downloading Data...");

                try
                {
                    //TimeNow = DateTime.Now.ToUniversalTime();
                    TimeNow = DateTime.Now;
                    mnth = TimeNow.Month.ToString();
                    date_ = TimeNow.Day.ToString();
                    year = TimeNow.Year.ToString();
                    hour_ = TimeNow.ToString("hh:mm:ss
tt").Split(":".ToCharArray())[0];
                    mn_ = TimeNow.ToString("hh:mm:ss
tt").Split(":".ToCharArray())[1];
                    ampm = TimeNow.ToString("hh:mm:ss tt").Split("
".ToCharArray())[1];
                    ed = "#" + mnth + "/" + date_ + "/" + year + " " + hour_ + ":"
+ mn_ + ":00 " + ampm + "#";

                    //TimeNow = DateTime.Now.AddMinutes(-
1*N*60).ToUniversalTime();
                    TimeNow = DateTime.Now.AddMinutes(-1 * N * 60);
                    mnth = TimeNow.Month.ToString();
```

```
                    date_ = TimeNow.Day.ToString();
                    year = TimeNow.Year.ToString();
                    hour_ = TimeNow.ToString("hh:mm:ss
tt").Split(":".ToCharArray())[0];
                    mn_ = TimeNow.ToString("hh:mm:ss
tt").Split(":".ToCharArray())[1];
                    ampm = TimeNow.ToString("hh:mm:ss tt").Split("
".ToCharArray())[1];
                    st = "#" + mnth + "/" + date_ + "/" + year + " " + hour_ + ":"
+ mn_ + ":00 " + ampm + "#";

                    str = "SELECT BTMacs.Sender, BTMacs.MAC, BTMacs.TSTamp FROM
BTMacs " +
                        "WHERE ( ((BTMacs.Sender)=" + up + ")" +
                        " AND (([BTMacs]![TSTamp]) Between " + st + " And " + ed +
"));";
                    OleDbCommand Cmd = new OleDbCommand(str, connection);

                    using (OleDbDataReader reader = Cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            string mcad =
reader["MAC"].ToString().Split(",".ToCharArray())[0];
                            string tsp = reader["TSTamp"].ToString();
                            if (BTData[up].ContainsKey(mcad))
                            {
                                BTData[up].Remove(mcad);
                                BTData[up].Add(mcad, tsp);
                            }
                            else
                                BTData[up].Add(mcad, tsp);
                        }
                    }

                    str = "SELECT BTMacs.Sender, BTMacs.MAC, BTMacs.TSTamp FROM
BTMacs " +
                         "WHERE ( ((BTMacs.Sender)=" + down + ")" +
                         " AND (([BTMacs]![TSTamp]) Between " + st + " And " + ed
+ "));";
                    Cmd = new OleDbCommand(str, connection);

                    using (OleDbDataReader reader = Cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            string mcad =
reader["MAC"].ToString().Split(",".ToCharArray())[0];
                            string tsp = reader["TSTamp"].ToString();
                            if (BTData[down].ContainsKey(mcad))
                            {
                                BTData[down].Remove(mcad);
                                BTData[down].Add(mcad, tsp);
                            }
                            else
```

```
                                BTData[down].Add(mcad, tsp);
                            }
                        }

                    }
                    catch (OleDbException)
                    {
                        Console.WriteLine("Error while uploading the data");
                    }

                    connection.Close();

                    Console.WriteLine("Estiamting ravel Time...");
                    sw.Write(DateTime.Now);

                    if (BTData[up].Count > 0 && BTData[down].Count > 0)
                    {
                        foreach (string macadd in BTData[down].Keys)
                        {
                            if (BTData[up].ContainsKey(macadd))
                            {
                                upts = ConvertTime(BTData[up][macadd]);
                                dnts = ConvertTime(BTData[down][macadd]);

                                sw.Write("," + (dnts - upts).ToString());
                            }

                        }
                        sw.WriteLine();
                    }
                    else
                        sw.WriteLine("No Data Collected");

                    sw.Flush();

                    System.Threading.Thread.Sleep(N*60*1000); // update interval of N
minutes
                }
            }

            private static int ConvertTime(string date_)
            {
                int timeinsecond = 0;
                int yy_;
                int mn_;
                int dd_;
                int hh_;
                int mm_;
                int ss_;
                string tt;

                string[] tstamp = null;

                mn_ = int.Parse(date_.Split("/".ToCharArray())[0]);
                dd_ = int.Parse(date_.Split("/".ToCharArray())[1]);
```

```
            //yy_ = int.Parse(date_.Split("/".ToCharArray())[2]);

            tstamp = date_.Split(" ".ToCharArray())[1].Split(":".ToArray());

            hh_ = int.Parse(tstamp[0]);
            mm_ = int.Parse(tstamp[1]);
            ss_ = int.Parse(tstamp[2]);

            tt = date_.Split(" ".ToCharArray())[2];

            if (tt == "PM")
                hh_ = hh_ + 12;

            timeinsecond = ss_ + 60 * mm_ + 3600 * hh_ + 3600 * 24 * dd_;

            return timeinsecond;
        }
    }
}
```