Coordination of Autonomous Vehicles at Intersections with Decentralized V2V Communication

Maite González Mendoza[1,3], Benjamín Holloway[1,3], Alejandro Hevia[1,3], Sandra Céspedes[1,2], Javier Bustos[1,3]
{maite, benjamin, scespede, ahevia, jbustos}@niclabs.cl
NIC Chile Research Labs[1]
Departamento de Ingeniería Eléctrica[2]
Departamento de Ciencias de la Computación[3]
Universidad de Chile

Resumen

In order to minimize the accidents, autonomous vehicles are being designed as the future of transportation. To fulfill this goal, we should look at vehicular intersections, where unfortunately most car accidents take place. In "Automation of a T-intersection using virtual platoons of cooperative autonomous vehicles", the algorithm called "Target Vehicle Assignment(TVA)" was proposed to resolve the problem of deciding an order in which vehicles cross in a 4-way intersection. This article reviews this algorithm and uses it as starting point to propose novel algorithms that fix issues detected in the original algorithm. The newly proposed algorithms are designed to be used with Vehicle To Vehicle (V2V) communications and are descentralized to avoid dependency of specific hardware in any intersection.
Keywords: Autonomous vehicles, autonomous intersection management, vehicular intersection control algorithms.

1. Introducción

Cuando se tiene una intersección y existen varios vehículos que quieren cruzar, se debe coordinar de forma que los vehículos que crucen la intersección al mismo tiempo no crucen sus trayectorias. Hoy en día comúnmente se utilizan señaléticas o semáforos para realizar esta coordinación, pero en el futuro próximo se espera que los automóviles se comuniquen entre sí, para establecer una coordinación mediante algoritmos como los presentados en [1, 2].En el caso en que los vehículos cruzan sus trayectorias, se debe encontrar un orden donde primero cruza uno y luego el otro. En este documento analizaremos el algoritmo propuesto en [1] y se proponen otros algoritmos para mejorar ciertos aspectos.
Si los vehículos se pueden comunicar dentro de los márgenes de una zona de cooperación (CZ del inglés *Cooperation Zone*) entonces se pueden definir algoritmos de coordinación que tomen información de los vehículos (como posición, velocidad, vehículos existentes en la CZ, etc), y definir cierto orden que asegure una intersección libre de colisiones.

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

2. Algoritmos Básicos

2.1. Algoritmo con Caravanas Virtuales (CV)

El primer algoritmo a analizar es el algoritmo de Asignación de vehículo objetivo (*Target Vehicle Assignment*) [1]. En este algoritmo, para decidir el orden de cruce de una intersección, a cada vehículo nuevo v_i que ingresa a la CZ le asigna un vehículo objetivo a seguir, utilizando información de v_i tal como su carril de entrada y su intención (virar a la derecha o izquierda, o seguir derecho), generando así pelotones virtuales.
El proceso está descrito en pseudocódigo en el Algoritmo 1.

Data: v: Vehículo nuevo, L: Lista de vehículos viejos.
Result: id: Identificador del vehículo a seguir.
$ordenaPorTpoIngreso(L)$;//más nuevo al más viejo
for $v' \leftarrow L$ **do**
  if $cruzanCaminos(v, v')$ **then**
    return $id(v')$
  end
end

Algoritmo 1: Algoritmo TVA con caravanas [1]

Para ilustrar el funcionamiento de este algoritmo, consideremos el escenario del cuadro 1, en una intersección en cruz con una vía en cada sentido. Los vehículos V_1 al V_5 ingresan y se procesan en el orden que aparece

Id	Origen	Destino		Vehículos	Objetivo
V_1	Norte	Sur	$\downarrow$	{}	-
V_2	Oeste	Este	$\rightarrow$	$\{V_1\}$	V_1
V_3	Sur	Este	$\uparrow$	$\{V_2,V_1\}$	V_2
V_4	Este	Oeste	$\leftarrow$	$\{V_3,V_2,V_1\}$	V_1
V_5	Norte	Este	$\downarrow$	$\{V_4,V_3,V_2,V_1\}$	V_4

Cuadro 1: TVA según [1]

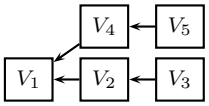

Figura 1: Caravana formada con el algoritmo original

en la tabla. La flecha muestra de manera gráfica el origen y destino del vehículo. En la columna "Vehículos" se ven los vehículos que están en la intersección al momento de ingresar a la CZ, y en la columna "Objetivo" se indica el vehículo que asignado según el Algoritmo 1. La Figura 1 muestra la caravana obtenida en este ejemplo. Aquí se aprecia que los vehículos V_3 y V_5 se encuentran caravanas distintas y en el mismo nivel, por lo que según el algoritmo descrito deberían poder pasar al mismo tiempo. Sin embargo, en el caso del ejemplo, estos cruzan trayectorias lo que conllevaría a una colisión.

En el trabajo donde se propone este algoritmo, se utilizan técnicas de *collission avoidance* para evitar estas colisiones. Sin embargo, en nuestra opinión, una mejor estrategia es diseñar métodos de asignación de cruce que eviten esto desde el inicio.

2.2. Algoritmo con CV Mejorado

En vista del que el algoritmo anterior provoca colisiones, una posible mejora consiste en ordenar los vehículos de manera distinta, como se muestra en el algoritmo 2. Comparado con el algoritmo 1, aquí se agrega el factor de orden por profundidad en la caravana.

Data: v: Vehículo nuevo, L: Lista de vehículos viejos en la intersección
Result: id: Identificador del vehículo a seguir
$ordenaPorTpoIngreso(L)$;//más nuevo al más viejo
$ordPorProfEnCarav(L)$;//de + a − profundidad
for $v' \leftarrow L$ **do**
  if $cruzanCaminos(v, v')$ **then**
    return $id(v')$
  end
end

Algoritmo 2: Algoritmo TVA [1] Mejorado

Con esa mejora, para el ejemplo anterior se obtiene una simulación como la que se ve en el cuadro 2. La única diferencia está en que V_5 debe seguir ahora el vehículo V_3, puesto que al calcular el nuevo orden, es el primer vehículo con el cual choca V_5. Con esto la caravana queda como se ve en la Figura 2, evitando así la colisión que se presentó con el algoritmo anterior.

Id	Origen	Destino		Vehículos	Objetivo
V_1	Norte	Sur	$\downarrow$	{}	-
V_2	Oeste	Este	$\rightarrow$	$\{V_1\}$	V_1
V_3	Sur	Este	$\uparrow$	$\{V_2,V_1\}$	V_2
V_4	Este	Oeste	$\leftarrow$	$\{V_3,V_2,V_1\}$	V_1
V_5	Norte	Este	$\downarrow$	$\{V_3,V_4,V_2,V_1\}$	V_3

Cuadro 2: TVA con la primera mejora

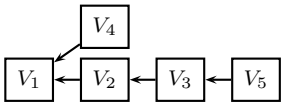

Figura 2: Caravana formada con la primera mejora

3. Efectos de la mejora

Los resultados que se ven en la Figura 3 dejan ver que la primera mejora disminuyó considerablemente la cantidad de colisiones, sobre todo cuando se aumenta la distancia mínima que tiene que haber entre los vehículos. Sin embargo, las colisiones no fueron eliminadas por completo.

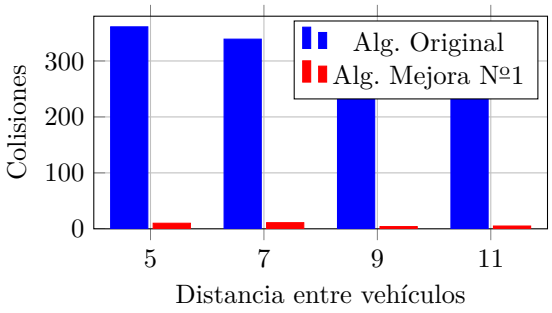

Figura 3: Cantidad de colisiones variando distancia

4. Nuevos Algoritmos

Dado que la mejora propuesta no soluciona el problema de colisiones, se proponen nuevos algoritmos que ahora toman en cuenta otros factores, tales como la velocidad del vehículo que ingresa, su posición, su distancia hasta la intersección y su tiempo dentro de la intersección.

Cabe destacar que si tenemos los vehículos de la Figura 4, podemos saber que V_1 cruza trayectoria con V_2 y V_3, pero que entre ellos no cruzan trayectorias. Sin embargo, no podemos decir nada con respecto a la relación de cruce que existe entre V_2 y V_4 ó V_1 y V_4, ya que las relaciones descritas por "chocar" o "no chocar" no son transitivas.

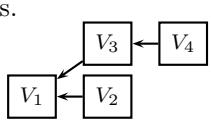

Figura 4: Ejemplo de caravana

4.1. Propuesta 1: Caravanas FIFO sin colisión

Este algoritmo se basa en el anterior, pero además revisa si hay colisiones en las N potenciales caravanas. La descripción del algoritmo 3 asume que existe un árbol que guarda la información de las caravanas (vehículos ya asignados, jerarquía de seguimiento y tiempo de cruce asignado de cada vehículo).

Data: v: Vehículo nuevo, AC: Árbol de caravanas
Result: vf: vehículo a seguir
$vehsQueCruza \leftarrow$; **for** $Carav \leftarrow AC$ **do**
  for $v' \leftarrow Carav$ **do**
    if $cruzanCaminos(v, v')$ **then**
      $vehsQueCruza.append(v')$; break;
    end
  end
end
$vf \leftarrow vehMaxTpoDeCruce(vehsQueCruza)$;
$v.tpoCruce \leftarrow calcTpoCruce(v)$; **return** vf

Algoritmo 3: Algoritmo de caravanas FIFO

Si bien no lo hemos demostrado formalmente, conjeturamos que este algoritmo está libre de colisiones. Un potencial problema es su eficiencia en el peor caso: cuando el vehículo que ingresa no cruza trayectoria con ningún otro, se debe comparar con todos los vehículos en la CZ. Dado que esto ya ocurre en el caso anterior (cuando hay sólo una caravana y no cruza trayectoria con ningún otro vehículo) su impacto es menor. Por otro lado, este algoritmo es conservador y asegura un orden FIFO, lo cual no siempre es óptimo. Concretamente, obliga a vehículos a siempre esperar a que el que entró primero salga primero, aun cuando podrían haber cruzado sin riesgo a colisión.

4.2. Propuesta 2: Bloques de Tiempo

A fin de optimizar el algoritmo anterior en relación al tiempo de espera de los vehículos, permitimos a los vehículos ingresar en un orden no FIFO y así optimizar los tiempos de cruce y de espera. Se definen dos propiedades de los vehículos: origen y dirección. El primero es el lugar de donde viene el vehículo: Norte, Sur, Este u Oeste. La dirección que corresponde a la intención:"seguir derecho", "doblar a la derecha", "doblar a la izquierda" o "todos"[1]. Esto entrega 16 combinaciones distintas. La estructura de datos para este algoritmo consiste en 16 arreglos de tipo *bool*, uno para cada combinación. Cada arreglo asociado a una tupla <origen, dirección> representa el tiempo dividido en bloques, y el bloque $k-$ésimo es verdadero si existe un vehículo que intenta cruzar la intersección en el tiempo k con el origen y destino indicados. El Algoritmo 4 define esta propuesta.

[1]La dirección "todos" se reserva para los casos donde hay algún vehículo en el mismo carril (mismo origen) y así poder calcular el tiempo mínimo de llegada tomando en cuenta vehículos que antecedan al vehículo que va ingresando.

Data: v: Vehículo nuevo, $AB[ori][dir][tpo]$: Arreglos de bloques de tiempo
Result: bs: bloques de tiempo para cruzar
$tml \leftarrow tpoMinLlegadaIntersect(v)$;
$ti \leftarrow getTpoInterseccion(v)$;
$arrsCol \leftarrow getArreglosColision(v)$;
$arrCol \leftarrow mergeArreglosColision(arrsCol)$;
$kBloques \leftarrow getBloquesDisponibles(arrCol, v)$;
for $k \leftarrow kBloques$ **do**
  $AB[v.ori][v.dir][k] \leftarrow true$;
  $AB[v.ori][\text{"todos"}][k] \leftarrow true$;
end
return $kBloques$

Algoritmo 4: Algoritmo de bloques de tiempo

El tiempo mínimo de llegada a la intersección corresponde al tiempo que demora el vehículo en llegar hasta la intersección tomando en cuenta factores como la velocidad máxima y los vehículos que lo anteceden. Por otro lado, la función *getArreglosColision* retorna a todos los arreglos $AB[orig'][dir']$ que correspondan a orígenes y direcciones que crucen la intersección con el vehículo que va ingresando. La función *mergeArreglosColision* realiza un OR ("o lógico") entre los *bool*, así obteniendo un sólo arreglo donde hay $True$ sólo en los bloques donde pasan vehículos con los cuales se colisiona.

Por la forma en que se asignan los bloques de tiempo-espacio, conjeturamos que el algoritmo propuesto es *optimal greedy*.

5. Conclusiones

El algoritmo original presentado por Medina2015 [1] presenta falencias relacionadas con las colisiones entre vehículos en las implementaciones desarrolladas en nuestro trabajo. Si bien con el primer arreglo se lograron eliminar la mayoría de las colisiones existentes, no fueron completamente eliminadas. Con las soluciones propuestas se pretende eliminar todas las colisiones, aunque todavía se requiere una verificación teórica y experimental para obtener resultados definitivos. Conjeturamos que es posible optimizar la última solución propuesta a fin de alcanzar un algoritmo correcto *optimal greedy*.

6. Trabajo Futuro

En un futuro trabajo se buscará verificar la correctitud de manera teórica y experimental de los algoritmos propuestos. Para ello, se utilizará tanto en el *framework* especializado diseñado Holloway16 [3, 4] como en el *framework Veins* [5]. Este último utiliza *Sumo* para el tráfico vehicular y *Omnet++* para la simulación de redes. Además, conjeturamos que el último algoritmo propuesto tiene potencial para ser *optimal greedy*, lo cual requiere ser probado formalmente.

Agradecimientos

Este proyecto ha sido financiado a través del NIC Chile Research Labs, el proyecto "RETRACT" ELAC 2015/T10-0761 y el Instituto de Sistemas Complejos de Ingeniería, ISCI (CONICYT: FB0816).

Referencias

[1] A. I. M. Medina, N. V. D. Wouw, and H. Nijmeijer, "Automation of a T-intersection Using Virtual Platoons of Cooperative Autonomous Vehicles," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2015-Octob, pp. 1696–1701, 2015.

[2] M. Elhadef, "An adaptable invanets-based intersection traffic control algorithm.," in *CIT/IUCC/DASC/PICom* (Y. Wu, G. Min, N. Georgalas, J. Hu, L. Atzori, X. Jin, S. A. Jarvis, L. C. Liu, and R. A. Calvo, eds.), pp. 2387–2392, IEEE, 2015.

[3] B. Holloway, "T-intersection algorithm implementation and attacks." `https://github.com/bhollowa/t-intersection-implementation`, 2016–2017.

[4] B. Holloway, S. Céspedes, and A. Hevia, "Analisys of attacks to automated vehicular coordination systems at intersections," 2016.

[5] F. Dressler, C. Sommer, D. Eckhoff, and O. K. Tonguz, "Towards Realistic Simulation of Inter-Vehicle Communication: Models, Techniques and Pitfalls," *IEEE Vehicular Technology Magazine*, vol. 6, pp. 43–51, September 2011.