차량용 레이더 센서를 위한 다중 타겟 알고리즘의 구현과 평가

Implementation and Evaluation of Multiple Target Algorithm for Automotive Radar Sensor

유 인 환* · 원 인 수** · 권 장 우***

* 주저자 : 인하대학교 컴퓨터공학과 석사과정
** 공저자 : 인하대학교 정보전자공동연구소 연구교수
*** 교신저자 : 인하대학교 컴퓨터공학과 교수

In-hwan Ryu* · In-Su Won** · Jang-Woo Kwon***

* Dept. of Computer Science and Information Engineering, Inha Univ.
** Institute for Information and Electronics Research, Inha Univ.
*** Dept. of Computer Science and Information Engineering, Inha Univ.

† Corresponding author : Jang-Woo Kwon, jwkwon@inha.ac.kr

Vol.16 No.2(2017)
April, 2017
pp.105~115

ISSN 1738-0774(Print)
ISSN 2384-1729(On-line)
https://doi.org/10.12815/kits.2017.16.2.105

Received 6 March 2017
Revised 23 March 2017
Accepted 23 March 2017

요 약

루프 검지기, 영상 검지기 등의 기존 교통 검지기들은 설치와 유지보수에 드는 비용이 크고, 밤과 낮에 따라 상이한 검지 알고리즘이 필요하거나 날씨에 따라 검지율의 편차가 크다는 단점을 가지고 있다. 반면에 밀리미터파 레이더는 악천후에 의한 영향을 받지 않고, 주야간에 관계없이 일정한 검지 성능을 얻을 수 있다. 덧붙여 설치와 유지보수를 위하여 교통 통제의 필요가 없고, 다수의 차량을 동시에 검지 가능하다. 본 연구는 이러한 장점을 가진 레이더 센서를 활용한 다중 물체 검지 알고리즘을 기존의 단일 물체 검지 알고리즘을 응용하여 구현하였으며 이에 대한 평가를 수행하여 의미 있는 결과를 얻었다.

핵심어 : 다중 물체 추적, 차량 검지, 과속 단속, 레이더, 지능형 교통 시스템

ABSTRACT

Conventional traffic detection sensors such as loop detectors and image sensors are expensive to install and maintain and require different detection algorithms depending on the night and day and have a disadvantage that the detection rate varies widely depending on the weather. On the other hand, the millimeter-wave radar is not affected by bad weather and can obtain constant detection performance regardless of day or night. In addition, there is no need for blocking trafficl for installation and maintenance, and multiple vehicles can be detected at the same time. In this study, a multi‐target detection algorithm for a radar sensor with this advantage was devised / implemented by applying a conventional single target detection algorithm. We performed the evaluation and the meaningful results were obtained.

Key words : multi-target tracking, vehicle detection, speeding detection, radar sensor, intelligent transport system

Ⅰ. 서 론

1. 개 요

다양한 종류의 장비들이 교통량 측정에 쓰이고 있다. 이와 같은 측정 장비의 예로 닫힌 고리 형태의 안테나를 도로 포장재 안에 묻어 차량이 그 위를 지나갈 때의 전자기유도를 이용하여 차량의 속도를 알아내는 루프 검지기, 구간 속도 측정을 위하여 입·출구에 설치되는 영상 검지기 등이 있다. 하지만 루프 검지기는 설치를 위하여 교통을 통제하고, 도로를 파헤쳐야한다는 문제점을 가지고, 영상 검지기 등은 날씨, 광량 등에 따라 검지율에 확연한 차이가 생기는 단점을 가지고 있다(Kang et al., 2002). 레이더 센서는 이러한 기존 교통량 측정 장비들의 단점을 해결할 수 있는 센서이다. 수십 GHz 대의 전파를 사용하는 레이더는 날씨의 영향을 받지 않고(Honma and Uehara, 2001), 루프 검지기와 같이 설치와 점검을 위하여 교통을 통제할 필요도 없다. 덧붙여, 지능형 자동차의 보급 확대로 값싼 차량용 레이더 센서가 출시되고 있어 비용 측면에서도 기존의 센서들과 견줄만 하다. 본 연구는 이러한 장점을 가진 저가의 차량용 레이더 센서 1기를 이용하여 다수의 차량 속도를 측정하고, 여러 차선을 지나는 교통량을 동시에 측정하기 위하여 단일 타겟 추적 알고리즘인 PDAF[1](Bar-Shalom and Tse, 1975)를 이용하여 다중 타겟 추적에 이용할 수 있는 스킴(scheme)을 고안하여 레이더 센서를 위한 다중 타겟 추적 교통량 측정 시스템을 고안하고 평가하였다.

Ⅱ. 단일 타겟 추적 알고리즘

단일 타겟 추적 알고리즘인 PDAF는 하나의 타겟을 추적하는 데 하나의 측정치만을 반영하는 칼만필터(Kalman Fileter)와는 달리 다수의 측정치를 하나의 타겟의 추적에 이용한다. 이 때, 각 측정치가 타겟의 상태 추정에 기여하는 정도는 타겟의 이전 추정 위치와 각 측정치의 통계적인 거리인 마할라노비스 거리(Mahalanobis distance)에 비례한다. 본 논문이 고안한 스킴을 이해하기 위하여 고전적인 필터인 PDAF의 작동 과정을 다시 살펴보면 다음과 같다.

1. PDAF의 작동 과정

PDAF의 작동은 마지막으로 추정된 타겟 위치와 현재 측정된 모든 측정치 간의 마할라노비스 거리를 산출함으로써 시작된다. 이 거리가 역치(γ)보다 작은 경우에 한하여 해당 측정치를 PDAF의 갱신에 사용된다. 이렇게 선별된 측정치가 타겟의 갱신에 기여하는 정도를 나타내는 β를 모든 측정치에 대하여 계산한 후, 칼만 필터의 혁신(innovation)에 대응되는 개념인 합성 혁신(combined innovation)을 사용하여 타겟의 새 상태를 추정한다. 칼만필터와 마찬가지로 PDAF의 작동 과정은 크게 예측 단계, 보정 단계의 두 가지 단계로 나눌 수 있다.

1) 예측 단계

PDAF의 예측 단계는 칼만필터와 같다. 예측 단계에서는 추적중인 물체의 바로 다음 상태에 해당되는 공

1) PDAF : probabilistic data association filter, 확률적 데이터 연관 필터

분산을 구할 수 있다.

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{F}_k \hat{\mathbf{x}}_{k|k} \tag{1}$$

$$\hat{\boldsymbol{z}}_{k+1|k} = \mathbf{H}_{k+1} \hat{\mathbf{x}}_{k+1|k} \tag{2}$$

$$\hat{\mathbf{P}}_{k+1|k} = \mathbf{F}_k \mathbf{P}_{k|k} \mathbf{F}_k^{\top} + \mathbf{Q}_k \tag{3}$$

여기서 $\hat{\mathbf{x}}_{k+1|k}$는 추적중인 물체의 k시간의 정보를 이용하여 $k+1$시간의 예측된 상태, $\mathbf{F}_k$는 타겟이 따르는 동적 시스템 모델의 상태 천이 행렬, $\hat{\boldsymbol{z}}$는 상태 $\hat{\mathbf{x}}$에 대한 측정치, $\hat{\mathbf{P}}$는 상태 벡터 $\mathbf{x}$ 의 각 차원이 서로 어떠한 관계를 맺고 있는지를 나타내는 공분산 행렬, $\mathbf{H}$는 상태 벡터 $\mathbf{x}$를 측정 벡터 $\boldsymbol{z}$로 사상하는 측정 행렬, $\mathbf{Q}$는 상태 천이 과정에서의 잡음의 공분산으로, 잡음의 분포는 다변수 가우시안 분포 $N(0,\mathbf{Q}_k)$를 따른다고 가정한다.

혁신 공분산(innovation covariance) 역시 칼만필터의 것을 사용한다.

$$\mathbf{S}_{k+1} = \mathbf{H}_{k+1} \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}^{\top} + \mathbf{R}_{k+1} \tag{4}$$

여기서 측정 벡터 $\hat{\boldsymbol{z}}$의 측정 과정에 대한 대한 $\mathbf{R}_{k+1}$은 측정 잡음 공분산 행렬이다.

2) 보정 단계

추적 대상 물체의 새 측정치를 예측한 후, 이를 중심으로 유효 게이트를 계산한다. 유효게이트는 예측된 측정치 $\hat{\boldsymbol{z}}$로부터 같은 마할라노비스 거리에 속하는 측정 공간 상의 점들의 집합이며 이 점들이 이루는 영역 ν는 다음과 같이 쓸 수 있다.

$$\nu = \left\{ \boldsymbol{z} | \boldsymbol{z} - \hat{\boldsymbol{z}}^{\top} \mathbf{S}^{-1} (\boldsymbol{z} - \hat{\boldsymbol{z}}) \leq \gamma \right\} \tag{5}$$

여기서 γ는 역치이며, 마할라노비스 거리가 카이제곱 분포를 따르므로 역 누적 카이제곱 분포로부터 얻을 수 있다. 추적중인 물체에서 측정치가 $(1-\alpha)$의 확률로 게이트를 벗어날 때의 역치 γ는 식(6)으로부터 구하여진다(Biley et al., 2006).

$$P_G = P\left(\frac{d}{2}, \frac{\gamma}{2}\right) \tag{6}$$

여기서,

$$P(a,b) = \frac{1}{\Gamma(a)} \int_0^b e^{-t} t^{a-1} dt \tag{7}$$

으로, 불완전 감마 함수(incomplete gamma function)이다. 설정한 게이트 안에 속하는 모든 측정치를 해당 PDAF의 입력 측정치로 설정하여 각각이 보정 단계에 기여할 정도인 가중치 β를 계산한다. 유효 게이트 안의 i번째 측정치의 가중치 β_i는 식 (8)로 주어진다(Bar-Shalom et al., 2006).

$$\beta_i = \begin{cases} \eta P_D N[\mathbf{z}_{k,i}; \hat{\mathbf{z}}_{k|k-1}, \mathbf{S}_k] & i = 1, \cdots, m_k \\ \eta(1 - P_D P_G) & i = 0 \end{cases} \tag{8}$$

여기서 m_k는 유효 게이트 안의 모든 측정치의 개수, P_D는 검출 확률(probability of detection), P_G는 측정 중인 물체에서 기인한 실제 측정치가 게이트 안에 포함될 확률로 식 (6)으로 주어진다. λ는 측정 영역(부피) 안에 균등하게 분포하는 거짓 측정치의 밀도이다. η는 정규화 상수로 모든 가중치 β_i의 합의 역수이다. 측정치의 인덱스 i는 1부터 m_k까지 부여되며, $i = 0$인 경우는 아무 측정치도 PDAF의 갱신에 사용하지 않고, 순전히 관성만을 반영하여 추정을 갱신하는 것에 해당한다.

가중치의 계산을 마치면, 다음과 같이 새 상태를 추정한다.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \nu_k \tag{9}$$

여기서 $\mathbf{K}_k$는 칼만필터의 칼만 게인과 동일하며, v_k는 합성 혁신으로,

$$\nu_k = \sum_{i=1}^{m_k} \beta_i \nu_i \tag{10}$$

의 각 측정치에 대한 개별 혁신 ν_i의 가중 평균으로 계산되며, 개별 혁신 ν_i는,

$$\nu_i = \mathbf{z}_{k,i} - \hat{\mathbf{z}}_{k|k-1} \tag{11}$$

처럼 칼만필터의 경우와 동일하게 주어진다. 공분산 행렬은,

$$\mathbf{P}_{k|k} = \beta_0 \mathbf{P}_{k|k-1} + (1 - \beta_0)(\mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top) + \tilde{\mathbf{P}}_k \tag{12}$$

와 같이 주어지며(Bar-Shalom et al., 2011), 첫 번째 항 $\beta_0 \mathbf{P}_{k|k-1}$은 아무 측정치도 갱신 과정에 반영 안하는 경우, 두 번째 항 $(1 - \beta_0)(\mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top)$은 측정치를 갱신 과정에 반영하는 경우에 해당한다. 두 경우가 각각 β_0, $1 - \beta_0$의 비중을 갖는 것을 확인할 수 있다. 마지막 항은

$$\tilde{\mathbf{P}}_k = \mathbf{K}_k \left(\sum_{i=1}^{m_k} \beta_i \nu_i \nu_i^\top - \nu_k \nu_k^\top \right) \mathbf{K}_k^\top \tag{13}$$

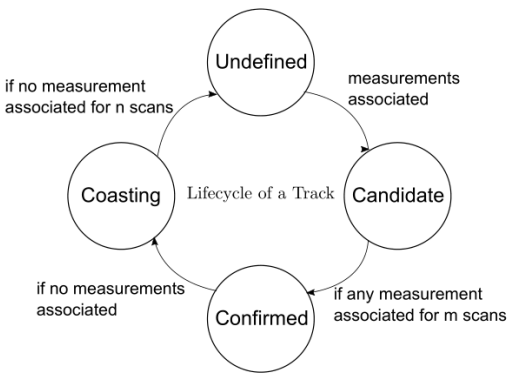
과 같으며, 측정치의 불확정성을 공분산에 반영한다(Bar-Shalom et al., 2009).

Ⅲ. 다중 물체 추적

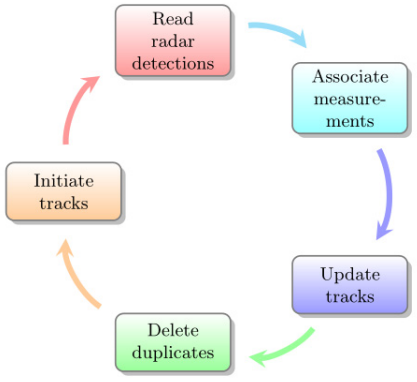
본 논문은 단일 물체 추적을 위한 알고리즘인 PDAF을 다중 물체 추적에 이용하기 위한 방법을 제안한다. 이를 위하여 각 측정치를 추적하는 추적기의 관리 스킴을 제시한다. 본 논문이 제안하는 다중 물체 추적기는 고정된 N개의 PDAF 단일 물체 추적기로 구성되어 있으며, 각 PDAF는 생애 주기에 따라 생성, 삭제, 초기화 등의 과정을 거친다. 다시 말해, <Fig. 1>과 같은 PDAF가 N개 있어, 추적기 관리 스킴이 각각의 생애주기를 관리하는 것이다.

1. 추적기의 생애 주기

<Fig. 1>은 각 추적기(PDAF)의 생애 주기를 나타낸다. '미정의(undefined)' 상태는 추적기에 아무 정보도 없음을 나타낸다. 측정치가 해당 추적기에 입력되면 '후보(candidate)' 상태로 천이한다. 후보 상태에 있는 추적기는 화면에 표시되지 않으며, 일정 시간 이상 지속시 '확정(confirmed)' 상태로 천이한다. 확정 상태에 있는 추적기의 출력은 화면에 출력되며, 단 한 순간이라도 아무 측정치도 배정되지 않을 경우 추적기의 상태는 '표류(coasting)'로 변한다. 표류 상태의 추적기 출력은 여전히 화면에 출력되지만, 이 상태가 일정 시간 이상 지속될 경우 '미정의(undefined)' 상태로 천이한다.



〈Fig. 1〉 Life-cycle of a track



〈Fig. 2〉 Proposed multi-target tracking algorithm

2. 추적기의 전체 구성

<Fig. 2>는 고정된 수의 단일 물체 추적기를 관리하는 추적기 관리 알고리즘(스킴)을 나타낸다. 레이더는 일정 이산시간마다 검지한 측정치 $\mathbf{z}$들의 리스트를 출력한다(read radar detections). 이어서 각 단일 물체 추적기의 게이트에 속하는 측정치를 배정하고(associate measurements), 해당 측정치들을 이용하여 각각의 추적기를 갱신한다(update tracks). 이어서, 중복된 추적기를 제거한다. 각 단일 물체 추적기는 서로의 존재를 알고 있지 않으므로 다수의 추적기가 동일한 물체를 추적할 가능성이 있고, 극단적으로는 모든 N 개의 추적기들이 모두 같은 물체를 추적하는 경우도 있을 수 있다. 이와 같은 현상을 방지하기 위하여, 모든 추적기 간의 유사도(similarity)를 계산하여 중복되는 추적기를 제거한다(delete duplicates). 새로 등장했지만 아무 추적기에도 속하지 못한 측정치는 빈(undefined 상태에 있는) 추적기를 하나 배정 받는다(initiate tracks).

3. 측정치 연관(Associate measurements)

〈List 1〉 List of Associate Measurements

```
1: procedure AssociateMeasurements(measurement measurements[])
2:     for all t in Trackers[ ] do
3:         x̂ = t.F · x
4:         P̂ = t.F · t.P · t.F^T + t.Q
5:         ẑ = t.H · x̂
6:         S = t.H · P̂ · t.H^T + t.R
7:         for all m in measurements[ ] do
8:             Δ = √((m.z − ẑ)^T · S^−1 · (m.z − ẑ))
9:             if Δ < γ then
10:                insert m into t's measurement list
11:                m.isAssociated =true
12:            end if
13:        end for
14:    end for
15: end procedure
```

<List 1>은 측정치 연관 알고리즘을 나타낸다. 추적중인 물체의 측정치 $\hat{\boldsymbol{z}}$를 계산하고, 현재 레이더가 읽어들인 모든 측정치와의 마할라노비스 거리를 계산하여 미리 설정한 역치 γ보다 작을 경우에 추적기의 측정치 리스트에 추가한다. 위 과정을 모든 N개의 추적기에 대하여 반복한다.

4. 추적 보정(Update tracks)

〈List 2〉 List of UpdateTracks

```
1: procedure UpdateTracks(measurement measurements[])
2:     for all t in Trackers[ ] do
3:         DoTransition(t)
4:         x̂ = t.F · t.x
5:         P̂ = t.F · t.P · t.F^T + t.Q
6:         ẑ = t.H · x̂
7:         S = t.H · P̂ · t.H^T
8:         K = P̂ · t.H · S^−1
9:         P^c = (I − K · H) · P̂
10:        if t.measNumber == 0 then
11:            insert ẑ into t's measurement list
12:        end if
13:        sum = 0
14:        for every measurement m in t's measurement list do
15:            m.ν = m.z − ẑ
16:            m.L = exp[−½ m.ν^T · S^−1 · m.ν]
17:            sum = sum + m.L
18:        end for
19:        L = 8 · (1 − P_G P_D) t.measNumber / P_D · γ^2
20:        sum = sum + L
21:        for every measurement m in t's measurement list do
22:            m.β = m.β / sum
23:            P̃ = P̃ + m.β · m.ν · m.ν^T
24:            ν^c = m.β · m.ν
25:        end for
26:        P̃ = K · (P̃ − ν * ν^T) · K^T
27:        t.P = L · P̂ + (1 − L) · P^c + P̃
28:        t.x = x̂ + K · ν^c
29:        add t.x into t's history list
30:        add L into t's β_0 list
31:    end for
32: end procedure
```

〈List 3〉 List of DoTransition

```
1: procedure DoTransition(t)
2:     if t.measNumber == 0 then
3:         if t.status == Candidate then
4:             t.statusTime = 0
5:             clear t's history list
6:             clear t's β_0 history list
7:             t.status = Undefined
8:         else if t.status == Coasting then
9:             t.statusTime = t.statusTime + 1
10:            if t.statusTime ≥ COAST_STEPS then
11:                t.statusTime = 0
12:                t.status = Undefined
13:            end if
14:        else
15:            t.status = Coasting
16:            t.statusTime = 0
17:        end if
18:    else
19:        if t.status == Candidate then
20:            t.statusTime = t.statusTime + 1
21:            if t.statusTime ≥ CANDIDATE_STEPS then
22:                t.status == Confirmed
23:                t.statusTime = 0
24:            end if
25:        else if t.status == Coasting then
26:            t.statusTime = t.statusTime + 1
27:            if t.statusTime ≥ COAST_STEPS then
28:                t.statusTime = 0
29:                t.status = Undefined
30:            end if
31:        else
32:            t.status = Coasting
33:            t.statusTime = 0
34:        end if
35:    end if
36: end procedure
```

모든 단일 물체 추적기는 <List 2>의 추적 보정 단계를 수행한다. 알고리즘은 우선 <List 3>의 상태 천이를 수행한다. 이어서, 추적 중인 물체의 예측치 $\hat{\mathbf{x}}$와 이와 연관된 공분산 $\mathbf{S}$, 칼만 게인 $\mathbf{K}$, 공분산 $\mathbf{P}^C$를 산출한다. 연관된 측정치 개수가 0일 경우($t.measNumber = 0$), 합성 혁신의 계산을 위하여 측정치 리스트에 예측치 $\hat{\mathbf{z}}$를 삽입하고, 가중치 β의 합 계산에 필요한 sum을 초기화 후, 모든 측정치의 혁신과 가중치를 계산한 후, sum에 더한다. 뒤이어 어떤 측정치도 참 측정치가 아닐 경우에 해당되는 가중치 β_0을 계산 후, 이 역시 sum에 더한다. 이 sum을 이용하여 각 측정치의 가중치 $m.\beta$를 계산하고 합성 혁신을 계산한다. 그 후, 추후에 중복 추적기를 제거하기 위해 이력 리스트에 새로이 추정된 상태 $t.\mathbf{x}$를 삽입하고 가중치 β_0을 넣는다.

<List 3>은 각 단일 물체 추적기의 상태 천이를 담당한다. 추적기에 입력된 측정치가 없고, 후보 상태인 경우 상태 유지 시간을 0으로 만든 뒤, 모든 이력을 삭제하고 미정의 상태로 바꾼다. 입력된 측정치가 없고, 표류 상태인 경우, 현재 상태가 유지되어온 시간인 statusTime에 1을 더한다. 더한 후의 값이 COAST_STEPS보다 크면, 미정의 상태로 천이한다. 측정치가 입력되었고, 후보 상태인 경우, statusTime에 1을 더하고, 그 값이 충분히 크다면 확정 상태로 천이한다. 측정치가 입력되었고, 표류 상태인 경우, statusTime에 1을 더하고 그 값이 COAST_STEPS를 넘어 서면, 미정의 상태로 천이한다.

5. 중복 추적기 제거

〈List 4〉 List of DeleteDuplicates

```
1: procedure DeleteDuplicates(t)
2:     Define matrix DeltaMat whose all elements are ∞
3:     sortedTrackList = sortByLifeTime(trackList)
4:     for i = 0 to TRACK_NUM −2 do
5:         for j = i + 1 to TRACK_NUM −1 do
6:             if sortedTrackList[i].status == Confirmed or
   sortedTrackList[j].status == Confirmed then
7:                 for t = 0 to HISTORY_NUM −1 do
8:                     sum = sum + √((x_{i,t} − x_{j,t}) S^{-1}_{j,t} (x_{i,t} − x_{j,t}))
9:                 end for
10:                DeltaMat[i, j] = sum / HISTORY_NUM
11:            end if
12:        end for
13:    end for
14:    for i = 0 to TRACK_NUM −1 do
15:        if sortedTrackList[i].status /= Confirmed then
16:            continue
17:        end if
18:        for j = 0 to TRACK_NUM −1 do
19:            if DeltaMat[i, j] ¡ γ then
20:                trackList[j].status = Undefined
21:            end if
22:        end for
23:    end for
24: end procedure
```

〈List 5〉 List of InitiateTracks

```
1: procedure InitiateTracks(t)
2:     for every track t in Trackers[ ] do
3:         if t.status == Undefined then
4:             add t to undefinedTracks[ ]
5:         end if
6:     end for
7:     for every measurement m in Measurements[ ] do
8:         if m is not associated with any of Tracks[ ] then
9:             add m to unAssocMeas[ ]
10:        end if
11:    end for
12:    if undefinedTracks is not empty then
13:        unAssocSize = number of measurements in unAssocMeas[ ]
14:        undefSize = number of measurements in undefinedTracks[ ]
15:        minSize = min(unAssocSize, undefSize)
16:        for i = 0 to minSize −1 do
17:            Tracker t = undefinedTracks[i]
18:            t.x = unAssocMeas[i].z
19:            t.P = c · Q_init
20:            t.state = Candidate
21:            t.lifeTime = 0
22:            t.statusTime = 0
23:        end for
24:        for i = unAssocSize to undefSize −1 do
25:            Tracker t = undefinedTracks[i]
26:            t.x = a random vector in observation volume
27:            t.P = c · Q_init
28:            t.state = Candidate
29:            t.lifeTime = 0
30:            t.statusTime = 0
31:        end for
32:    end if
33: end procedure
```

<List 4>는 마할라노비스 거리를 추적기 간 유사도의 척도로 이용하여 중복된 추적기를 제거한다. 알고리

즘은 우선 마할라노비스 거리를 저장하기 위한 행렬을 초기화하며 시작되어 모든 요소의 값을 무한대로 초기화한다. N개의 단일 물체 추적기를 저장하는 배열 trackList를 추적기의 나이인 lifeTime 내림차순으로 정렬하여 sortedTrackList에 저장한다. 그 후, 모든 추적기 쌍에 대하여 각 시간대끼리의 마할라노비스 거리를 구하고 이들의 평균을 구하여 DeltaMat에 저장하고, 이 행렬을 순회하면서 역치 γ 보다 작은 원소를 찾아 그 원소의 열 인덱스가 나타내는 추적기를 미정의 상태로 만든다.

6. 새 추적기 형성(Initiate tracks)

<List 5>가 나타내는 새 추적기 형성 단계는 미정의인 모든 추적기를 undefinedTracks 리스트에 삽입하고, 어떠한 추적기에도 추가되지 못한 측정치를 unAssocMeas 리스트에 삽입함으로써 시작된다. 어떠한 추적기에도 입력되지 못한 측정치들의 개수 unAssocSize와 미정의인 추적기의 개수 undefSize 중 최소값을 minSize에 입력함으로써 메모리 참조 오류를 방지할 수 있다. 이어서 unAssocMeas 리스트의 측정치를 하나씩 꺼내서 undefinedTracks에서 꺼낸 추적기에 입력한다. 둘 중 하나의 리스트가 비게 되면 위 과정이 종료되고, 만약 이 과정을 수행한 후에도 undefinedTracks 리스트가 비어있지 않다면 undefinedTracks 리스트의 모든 추적기의 상태를 측정 영역의 임의 점으로 정한다.

Ⅳ. 구현 및 평가

1. 구 현

구현을 위하여 Contintal사의 차량용 장거리 레이더 센서 ARS-308를 사용하였다. 본 센서는 77GHz 대역의 FMCW[2]파형을 사용하여 물체의 위치와 속도를 측정한다. 행렬 연산을 빠르게 처리하기 위하여 선형대수 라이브러리 Eigen을 사용하였고, 다중 물체의 추적을 실시간으로 확인하기 위하여 Qt 크로스플랫폼 프레임워크를 이용하였다. <Fig. 3>은 구현된 소프트웨어의 유저 인터페이스를 나타낸다.

1) 동적 시스템 모델

각 타겟의 움직임을 등속 모델(constant velocity model)을 따른다고 가정하였다. 등속 모델은

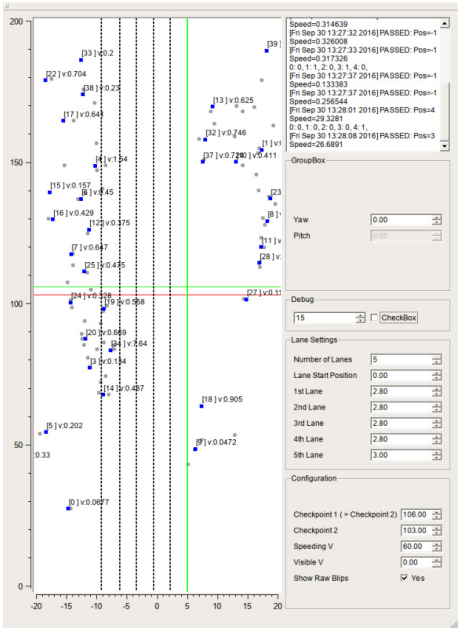
$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_k \tag{14}$$

와 같이 주어지며, 여기서,

$$\mathbf{x}_{k-1} = \begin{bmatrix} p_{x,k-1} \\ \dot{p}_{x,k-1} \\ p_{y,k-1} \\ \dot{p}_{y,k-1} \end{bmatrix} \tag{15}$$

2) FWCW : Frequency-Modulated Continuous Wave, 주파수 변조 연속파

$$\mathbf{F}_{\mathrm{k}}=\begin{bmatrix}1 & \Delta t & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & \Delta t\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{16}$$



〈Fig. 3〉 Graphical user interface of the implemented software

이며, $p_{x,k-1}$, $p_{y,k-1}$는 물체의 $k-1$시간에서의 x,y축의 위치 좌표, $\dot{p}_{x,k-1}$,$\dot{p}_{y,k-1}$는 속도이다. $\mathbf{w}_{\mathrm{k}}$는 $N(\mathbf{0},\mathbf{Q}_{\mathrm{k}})$를 따르는 잡음이다. 각 이산 시간의 간격이 66밀리초로 짧으므로 차량의 속도 차이는 무시할 수 있을 정도로 작다. 이러한 이유로 등속 모델을 사용하였다.

2) 사용 파라미터

본 연구가 사용한 파라미터는 다음과 같다.

$$\mathbf{F}_{\mathrm{k}}=\begin{bmatrix}1 & 0.066 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0.066\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{17}$$

$$\mathbf{H}_{\mathrm{k}}=\mathbf{I} \tag{18}$$

$$\mathbf{P}_{\mathrm{init}}=\mathbf{I} \tag{19}$$

$$\mathbf{Q}_{\mathrm{k}}=0.01\mathbf{I} \tag{20}$$

$$\mathbf{R}_{\mathrm{k}}=1.5\mathbf{I} \tag{21}$$

사상행렬 $\mathbf{H}_{\mathrm{k}}$는 측정 공간과 상태 공간이 같으므로 항등행렬을 사용하였고, 추적기의 초기화를 위한 초기

공분산 $\mathbf{P}_{init}$ 역시 항등행렬을 사용하였다. 프로세스 잡음 $\mathbf{Q}_k$는 매우 작은 값인 $0.01\mathbf{I}$, 측정 잡음 $\mathbf{R}_k$는 레이더 센서의 정확성을 고려하고, 시행착오를 통하여 $1.5\mathbf{I}$로 정하였다.

2. 평 가

본 연구 결과의 평가는 경인고속도로의 기점과 제2경인고속도로의 기점이 인접한 인천 남구 용현5동 낙섬사거리에서 실시하였다. 레이더를 향하여 다가오는 한 쪽 방향의 차량들에 대해서만 측정을 실시하였으며, 본래 차량의 전방 충돌 방지를 위하여 제작되어 방위 분해능에 한계를 가진 레이더 센서의 단점을 보완하기 위하여 1차로와 2차로, 4차로와 5차로의 통행량을 각각 한 개의 동일 차로로 간주하여 평가를 진행하였다.

1) 교통량 정확도

〈Table 1〉 Traffic measurement result of the proposed system (unit: # of cars)

	Actual				Measured			
	1st lane	2nd lane	3rd lane	Sum	1st lane	2nd lane	3rd lane	Sum
1st	25	15	25	65	24(4%)	15(0%)	28(12%)	67(3%)
2nd	27	22	40	89	33(22%)	25(14%)	48(20%)	106(19%)
3rd	28	13	33	74	23(18%)	12(8%)	35(6%)	70(5%)
4th	21	19	36	76	24(14%)	18(5%)	37(3%)	79(4%)
Sum	101	69	134	304	104(3%)	70(1%)	148(10%)	322(6%)

5분간 총 4회, 각 차로의 통행량을 계수하였다. 결과는 <Table 1>과 같다. 괄호 안은 상대오차를 나타낸다. 각 회의 차로별 교통량오차는 최대 22%인 것을 알 수 있으며, 모든 차로의 교통량을 더한 값은 상대오차가 최고 19%로, 비교적 작은 것을 알 수 있다. 다른 시행 보다 큰, 평균 19%의 상대오차를 보인 2차 시행은 적신호시 신호대기하는 차량들이 다른 시행에 비하여 두 배 이상 길었다. 이 때문에 차간 거리가 너무 좁아 레이더의 낮은 종방향 분해능으로는 구분이 어려웠고, 인접 차량과의 좁은 거리로 멀티패스(multipath)를 거친 전파가 소멸되지 않고 레이더로 도로 인입되는 등의 현상으로 큰 오차를 발생시킨 것으로 보인다. 이와 같은 현상을 개선하기 위해서, 분해능이 개선된 레이더 센서를 사용할 필요가 있다.

2) 속도 정확도

교통량 정확도 평가를 수행함과 동시에 속도 정확도 평가도 수행하였다. 레이더 센서가 설치된 육교로부터 100m 떨어진 지점에서부터 주행했을 때의 최고 속도와 본 연구의 시스템이 추정한 차량 속도를 비교하였다. 이 때, 실제 속도는 차량의 계기판에 표시된 값을 사용하였다.

〈Table 2〉 Comparison between actual speed and estimated speed of a car

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Actual Speed [km/h]	63	64	69	62	58	58	45	68	60
Estimated Speed [km/h]	58	60	69	59	54	59	45	89	59
Error	5	4	0	3	4	-1	0	-1	1
Relative Error (%)	7.9	6.3	0	4.8	6.9	1.7	0	1.5	1.7

<Table 2>는 실제 자동차의 속도와 본 연구가 구현한 추적기가 추정한 속도의 비교를 나타낸 것이다. 전체 9회 시행에 걸쳐 모두 10% 미만의 상대오차를 보이는 것을 확인할 수 있다.

Ⅴ. 결 론

본 연구는 높은 내구성, 유지보수 비용이 작은 특성 등을 가진 레이더 센서를 이용하여 지능형 교통 정보 수집 시스템을 구현하기 위하여 수행하였다. 한 대의 레이더 센서로 다차선을 동시 검지하기 위하여 다중 물체 추적 알고리즘을 널리 알려진 단일 물체 추적 알고리즘을 응용하여 고안하였고, 고안된 알고리즘을 이용하여 실제 도로 환경에서 시험한 결과, 교통량의 측정에서는 최대 22%의 상대 오차를, 속도 측정에서는 최대 8%의 상대오차를 보이는 것을 확인할 수 있었다. 교통량 측정에서의 다수 높은 오차는 각도 분해능이 더욱 뛰어난 센서를 사용하면 개선이 가능할 것으로 보이며, 시험 환경에 따라 각각의 단일 추적기(PDAF)에 쓰인 계수들을 정밀 조정하여 교통량과 속도 측정에서의 높은 정확도를 이끌어 낼 수 있을 것으로 예상된다. 이상의 개선 사항을 반영하여 레이더 센서의 높은 신뢰성을 바탕으로 유지보수 비용이 획기적으로 감소한 교통측정 시스템이 실현 가능할 것으로 예상된다.

ACKNOWLEDGEMENTS

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업 의 연구결과로 수행되었음"(IITP-2016-H8501-16-1019).

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. R7520-16-0005,작업자 및 사회적약자 맞춤형 근골격 안전시스템 구현을 위한 복합 3D 프린팅 활용 창의 기술 개발).

REFERENCES

Bar-Shalom Y. and Tse E.(1975), "Tracking in a cluttered environment with probabilistic data association," *Automatica*, vol. 11, no. 5, pp.451-460.

Bar-Shalom Y., Daum F and Huang J.(2009), "The probabilistic data association filter," *IEEE Control Systems,* vol. 29, no. 6, pp.82-100.

Bar-Shalom Y., Willett P. K. and Tian X.(2011), *Tracking and data fusion,* PBS Publishing.

Biley T., Upcroft B and Durrant-Whyte H.(2006), "Validation gating for non-linear non-Gaussian target tracking," *2006 9th International Conference on Information Fusion,* pp.1-6.

Honma S. and Uehara N.(2001), "Millimeter-wave radar technology for automotive application," *Signal,* vol. 1, pp.11-13.

Kang J. K., Son Y., Yoon Y. H. and Byun S.(2002), "Regional Traffic Information Acquisition by Non-intrusive Automatic Vehicle Identification," *The Journal of The Korea Institute of Intelligent Transport Systems,* vol. 1, no. 1, pp.22-32.