데이터마이닝을 통한 라인트레이서의 물리적 측정오류 패턴분석과 예측기법

김덕환[0] 이종욱 이찬근

중앙대학교 컴퓨터 공학과

gimdeokhwan@naver.com, peace0114@empal.com cglee@cau.ac.kr

Pattern Analysis and Predication of Physical Measurement Errors based on Data Mining for Line Tracers

Deok Hwan Gim[0] Lee Jong Uk Chan gun Lee

School of Computer Science and Engineering, Chung-Ang University 221 Heukseok, Dongjak, Seoul

요 약

실세계와 연동되는 컴퓨팅환경이 가속화 될수록 물리적 오류에 대한 중요성은 점차 커지고 있다. 물리적 오류는 자연현상과 밀접하게 관련되어 있기 때문에, 다양한 변수가 존재하며 오류 예측이 어렵다. 따라서 본 연구에서는 이러한 자연현상에서 발생하는 인자를 수집하고, 이를 분석하여 예측할 수 있도록 데이터마이닝을 적용한 시스템을 제안한다. 본 연구의 현실성 입증을 위해 제안한 시스템을 라인트레이서를 모델로 하여 구현해 보았다.

1. 서 론

IT분야가 항공, 기계 전기, 전자 등 인접 분야와 결합을 하면서 시스템은 점차 거대화 되어가고, 우리의 제어 관리는 점차 어려워지고 있다. 소프트웨어가 정상적 수행을 했음에도, 외부의 물리적 오류에 의해 잘못된 값이 입력으로 들어온다면, 정상적 수행이 오히려 시스템에 오류를 가져 오는 결과를 낳게 된다. 예를 들어 지하철 개찰구 시스템의 경우, 사람이 불법적으로 개찰구를 지나갈 때 반응하도록 설계 되어있다. 그러나 돈의 지불여부 또는 사람의 통과여부에 상관없이 외부 요인(바람, 빛 등)에 의해 개찰구가 닫히는 일이 간혹 발생한다. 이것은 소프트웨어가 정상적으로 동작했음에도 시스템은 비정상적으로 동작하는 명백한 오류로 볼 수 있다.

실세계와 연동되는 컴퓨팅환경이 가속화 될수록 물리적 오류에 대한 문제는 점차 가시화 될 것이다. 따라서 소프트웨어적인 오류만이 아닌 물리 환경에 따른 오류에 대해서도 안정화 로직이 필요하다. 그러나 이러한 물리적 오류는 그 원인을 밝혀내는 것이 쉽지 않기 때문에 오류의 패턴을 찾아내고자 한다. 그리고 찾아낸 패턴을 토대로 run-time시에 오류를 미연에 방지하는 시스템을 제안한다.

이러한 시스템을 구현하려면, 여러 실험을 통해 주변 환경에서 발생하는 물리적인 용인에 대한 데이터를 수집할 필요가 있다. 그리고 이 데이터를 이용하여 오류를 방지하기 위해서는, 오류 가능성을 예측하는 방법이 필요하다. 본 연구에서는 이를 위해 데이터 마이닝을 사용하였다.

본 논문에서 적용한 임베디드 시스템은 라인트레이서이다. 라인트레이서는 외부 빛의 영향에 매우 민감하다. 때때로 빛의 산란은 원치 않는 신호 검출의 원인이 된다. 이것은 정상적인 알고리즘 수행이 시스템의 오류를 가져오도록 유도한다. 따라서 이러한 환경에서 어떻게

오류를 찾아내며, 해결하는지에 대한 과정은 본 논문에서 제시하는 시스템의 현실성을 입증해줄 것이다.

본 논문의 구성은 다음과 같다. 2절에서는 라인트레이서에 대한 간략한 설명과 오류 가능성 예측 및 대처를 위한 관련연구로써 데이터 마이닝과 모니터링에서 대해서 설명한다. 그리고 3절에서는 실제 시스템 구현에 대해 구체적으로 기술한다. 4절에서는 구현된 환경에서의 실험데이터를 가지고 제시한 시스템이 어느 정도의 향상을 가져왔는지 비교 검증한다. 5절에서는 연구를 요약하고 향후 연구에 대해 다루며 결론 맺는다.

2. 관련 연구 및 기술

2.1 라인트레이서

라인트레이서는 바닥에 그려진 주행선을 센서로 검출해 목적 위치까지 이동하는 형태의 로봇으로 산업체에서 물건을 운반하는 무인차에 많이 응용되는 기술이기도 하다. 라인트레이서는 센서부에 해당되는 수발광센서와 액추에이터에 해당되는 조향, 속도 제어부로 나뉜다. 발광센서에서 라인을 향해 빛을 방출하면, 라인의 명도에 따라 서로 다른 양의 빛을 반사한다. 이 빛의 양은 수광센서에 의해 측정된다. 결과적으로, 측정된 빛은 라인의 위치를 탐색할 수 있게 해준다.

본 논문에서 사용된 라인트레이서 전용 프로세서는 Freescale(社)의 MC9S12 이다.[1] 통합개발환경인 코드워리어 내에 존재하는 bean expert 기능을 사용하면 라인트레이서에 필요한 여러 기능이 자동생성 된다.[2] 그림1 은 코드워리어 환경에서 bean expert를 사용하는 환경을 보여준다. 이것은 수작업을 통한 구현보다 오류를 줄일 수 있으며, 간편하게 코드를 작성할 수 있게 해준다

.

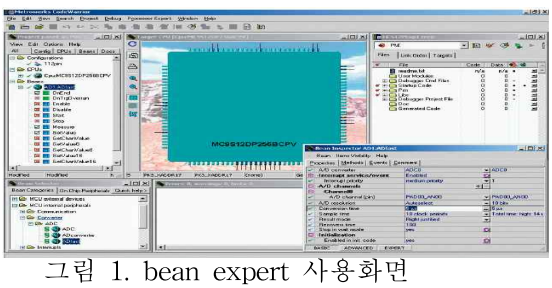

그림 1. bean expert 사용화면

2.2 데이터마이닝

데이터마이닝은 과거 데이터의 패턴을 분석하여 새로운 데이터와의 상관관계를 찾아내고 이를 토대로, 의미 있는 정보를 추출하기 위한 방법이다.[3][4] 데이터 마이닝 알고리즘은 여러 가지가 있지만 그중에서도 우리는 k-NN(k-Nearest Neighbor)알고리즘을 사용하였다.

k-NN은 예측이 필요한 새로운 데이터를 받고 이미 수집된 데이터(training set)들과 비교하는 방식으로 동작한다. 예측되는 데이터와 가장 유사한 상위 k개의 데이터를 찾고, 이 데이터들 중 다수를 차지하는 부류로 예측을 한다. 이 방식은 복잡한 함수안의 숫자를 예측하면서도 해석이 쉽다. 또한 데이터가 변경될 때마다 새로 학습시켜야 하는 방법과 달리 새로운 데이터를 언제든지 추가할 수 있다.[5]

2.3 모니터링

모니터링이란 시스템이 동작하는 동안에 시스템의 유효성이나 위반하면 안 되는 사항을 감지하는 기법이다.

데이터 마이닝을 통한 오류상황 예측을 위해서는 의미 있는 데이터를 추출해야 한다. 이것은 시스템이 동작하는 동안 물리적 오류가 발생할 수 있는 상황을 모니터링 함으로써 가능하다.[6]

본 논문에서는 라인트레이서가 블루투스를 통해 실시간으로 센서의 값을 원격의 PC에 설치된 모니터로 전송할 수 있게 구현하였다. 라인트레이서에 탑재된 프로세서의 성능상의 제약으로, 데이터마이닝은 PC에서 수행하였다.

3. 시스템 구현

3.1 시스템 구조 및 동작원리

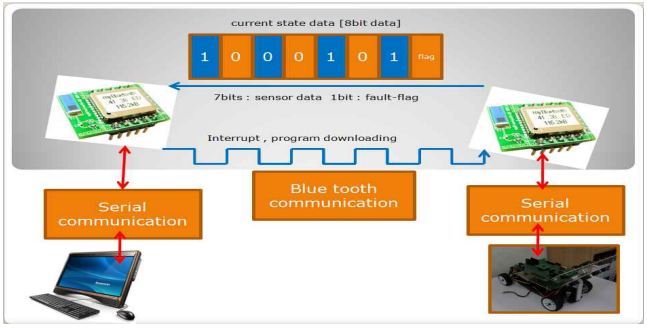

그림 2. 전체 시스템 구조

전체 시스템의 구조는 위의 그림2와 같은 형태로 되어있다. 전체적으로 1.블루투스 모듈부분 2.시리얼 통신부분 3. PC의 데이터처리부분 4.라인트레이서 부분으로 나눌 수 있다. 위의 시스템에서 라인트레이서에 탑재되어 있는 MC9S12라는 프로세서는 시리얼 통신을 지원한다. 따라서 시리얼 통신을 이용하여 PC와의 통신을 구현하였다. 그리고 Serial to Bluetooth 모듈을 사용하여 무선 환경에서 시리얼 통신이 가능하도록 구축하였다.[7]

라인트레이서는 각각의 센서 데이터 값을 시리얼 통신과 블루투스 모듈을 통하여 PC로 전송해준다. 모든 센서의 각 비트가 high를 나타내면, 라인을 이탈했다고 할 수 있다. 즉 모든 센서가 흰색을 가리킨다면 검정라인을 이탈한 것이다. 이것을 토대로 어떤 데이터부분에서 라인트레이서가 라인을 이탈했는지, 이탈하지 않았는지를 판단할 수 있다. 이것은 오류부분의 클래스와 오류가 일어나지 않은 클래스로 데이터를 나눌 수 있음을 의미한다. 누적된 데이터를 바탕으로 오류가 일어난 데이터 부근의 패턴을 분석하면 어떤 경우에 오류가 일어나는지를 판단할 수 있다. 데이터의 히스토리는, 우리가 라인트레이서의 패턴을 인식할 수 있게 도와준다. 이러한 작업은 상대적으로 고성능의 프로세서를 가진 PC에서 수행된다. 라인트레이서는 단지 데이터를 전송만 하고, PC에서 데이터 수집과 수집된 데이터를 이용한 데이터마이닝을 수행한다.

3.2. 시스템 수행과정

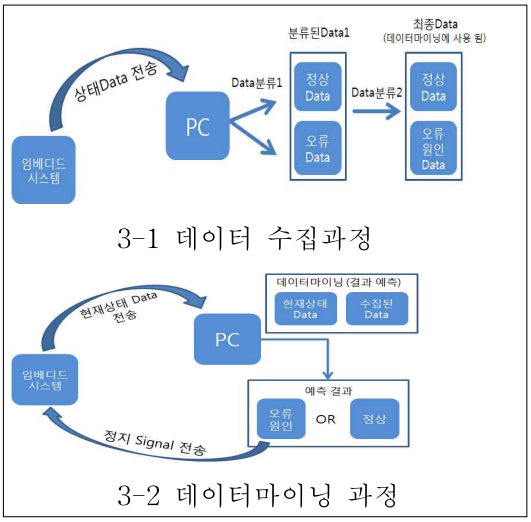

그림 3. 시스템 수행과정

구조를 알 수 없는 시스템을 해석 할 때, 입력으로 임펄스 신호를 인가하여 나오는 출력으로 시스템 내부를 알아낼 수 있다. 마찬가지로 원인을 알 수 없는 오류가 있을 때 오류에 의한 현상을 데이터 마이닝 하여 그 존재를 파악하고자 했다. 위의 그림3은 그 과정을 표현한 개념도이다. 그림3이 두 가지로 나누어 있는 것처럼, 이 과정 또한 두 가지로 나누어져 있다. 우선 데이터를 수집한다. 3-1과 같이 시스템의 상태 데이터를 수집한다. 이때 데이터는 오류일 때의 데이터와 정상 상태일 때의 데이터로 분류되어 저장된다. 그런데 우리가 실제로 필요로 하는 정보는 오류가 일어나기 직전의 상태 데이터 값이다. 따라서 정상상태의 데이터라 할지라도 오류 직전의 데이터라면 오류 원인 데이터로 분류되어 저장된다. 이것은 'Data분류2'를 통해 수행된다.

3-2는 그 데이터 수집 이후의 과정을 설명한다. 잘 분류되어 수집된 자료는 시스템의 미래 상태를 예측하는데 사용된다. 새로 들어온 데이터(현재 시스템의 상태)와 저장되어 있는 데이터를 비교하여 시스템이 미래에 오류가 날것인지, 아닌지를 판별한다. 그리고 그에 따른 트리거 신호를 발생시킨다. 모델로 제시한 라인트레이서 시스템의 경우, 오류 예측 지점에서 정지신호를 보냈다.

3.3 시스템 적용 시 유의점

```
오류 복구 시스템-1(현재 상태 데이터)
{

  if(오류 감지){
     if(현재  상태 데이터 == 왼쪽 코너링인 경우)
          오른쪽으로 코너링;
     if(현재  상태 데이터 == 오른쪽 코너링인 경우)
          왼쪽으로 코너링;
  }

}
```

그림 4. 의사코드

본 연구에서 제시하는 시스템은 데이터 마이닝을 이용하여 오류복구 동작을 수행한다. 그런데 여기에는 유의할 점이 있다. 왜냐하면 데이터마이닝이 100% 참인 경우를 예측하지 못하기 때문이다. 라인트레이서의 경우를 생각해보자. 데이터마이닝이 현재 상태를 오류로 예측했

다면, 현재 조향된 방향과 반대로 코너링을 해주어야 할 것이다. 그림4의 의사코드는 이것을 설명해 준다. 그러나 이것은 매우 위험할 수 있다. 데이터마이닝의 예측과 실제상황이 다르면, 잘못된 방향으로 코너링을 하여 라인 이탈을 유발할 수 있기 때문이다. 따라서 오류 가능성이 있다고 판단될 시에 멈추도록 구현하였다. 이것을 실생활에 적용하여 본다면 자동차 운전 시, 오류복구 동작을 수행 하는 것 보다는 운전자에게 경고 메시지를 보내는 것으로 대체할 수 있다. 즉, 이것은 보조 수단으로 활용되어야 한다.

4. 실험 및 검증

4.1 실험 및 검증 방법

라인트레이서가 오작동을 일으키는 잘 알려진 한 가지 케이스가 있다. 우리는 이 케이스를 인위적으로 일으켰다. 그리고 이 케이스를 얼마나 잘 검출해 내는지를 일반 라인트레이서와 비교, 검증하였다.

잘 알려진 케이스는 빛의 산란에 의한 라인 이탈이다. 라인트레이서가 코너링을 하면, 데이터는 센서에 순차적으로 들어온다. 만약 특정 센서를 건너뛰고 그 다음 센서로 데이터가 들어왔다면 이것은 빛의 의한 산란이 일어난 것이다. 우리가 빛의 산란을 일으키는 것은 어렵지만 센서 값을 조작하여 동일한 상황이 일어난 것처럼 환경을 조성하는 것은 어렵지 않다.

```
        for(i=0;i<8;i++){
        adc_cal_data[i] = 100*(adc_data[i] - adc_min[i]);
        position_temp[i] = adc_cal_data[i] / adc_diff[i];
        if((position_temp[i]&0xFF80) != 0)
            position_temp[i] = 100;
    }
    static int check_count = 0;
            if(check_count == 1000)
            {
                    position_temp[6] = 100;
                    position_temp[7] = 100;
                    position_temp[8] = 0;
            }

    for(i=0;i<8;i++)
        car_position_temp += (position_temp[i]*weight[i]);
    car_position = car_position_temp;
}
```

그림 5. 인위적 환경을 일으키도록 조작한 코드

위 그림은 라인트레이서 코드 중 일부를 발췌한 것으로 정상코드에 일부코드를 삽입하여 인위적으로 오류 상황이 발생하도록 만든 프로그램 이다. 위의 코드에서 작은 네모를 제외한 부분은 센서로부터 데이터를 입력받아 정규화 시키는 코드이며 이곳에 작은 네모의 코드를 추가시켜 센서 값을 인위적으로 조작하게 하였다. 작은 네모부분을 살펴보면 루틴내부에 static 변수를 삽입하여 일정시간이 흘렀을 때 실행되도록 하였다. 1000이 카운트 되었을 때 라인트레이서는 왼쪽으로 코너링을 해야 한다. 그러나 삽입된 코드는 position_temp[7]에 최솟값인 '0'을 입력한다. 이 변수에 최솟값이 입력되면 라인트레이서는 오른쪽으로 코너링을 해야 한다고 인식하게 된다. 그러나 position_temp[5], position_temp[6]에는 '100'이 인가되기 때문에 에러라고 간주할 수 있어야 한다. 작은 네모박스 안의 코드는 한번 주행 시 한번 수행되었다. 즉 한 번의 주행마다 한 번씩 멈추면 오류가 검출된 것으로 간주했다.

4.2. 실험 및 검증 결과

라인트레이서는 빛에 민감하므로 낮 과 밤 중 언제 수행되는가에 따라 많은 차이가 난다. 따라서 일몰 전 50번 일몰 후 50번으로 나누어 총 100번을 수행하였다. 본 연구에서 제안한 시스템을 적용한 라인트레이서는 100번의 실험이 수행 되었을 때 총 52번을 오류로써 감지해냈다.(일몰 후 50번 36번 성공, 일몰 전 50번 중 16번 성공) 제안된 시스템이 적용되지 않은 일반 라인트레이서의 경우 100번 수행시 100번 모두 라인을 이탈하였다. 아래의 그림은 실험 결과를 보여준다. 다른 물리적 상황에서도 동일한 예측성을 가져온다고 할 수는 없지만, 최소한 제안된 케이스 에서는 52%의 적중률을 보였다. 그러나 예상대로 일몰 전과 일몰 후의 성능차이는 극명하게 나타났다.

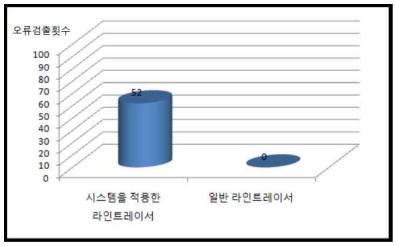

그림 6. 실험 결과

오류를 감지한 52개의 실험 데이터는 감지하지 못한 48개의 데이터와 달리 대부분, 비슷한 값의 분포를 보였

다. 유클리디안 거리 값도 3700~5400 사이의 값을 벗어나지 않았으며 오류클래스 데이터와 정상클래스 데이터의 개수 차이만 있었을 뿐, 순위는 항상 오류클래스의 데이터가 높았다. 52개의 데이터는 각각의 데이터 간의 차이가 거의 없기 때문에, 지면관계상 그중 하나의 데이터를 대표로 선택하여 보여주도록 하겠다.

s1	s2	s3	s4	s5	s6	s7	s8
91	60	28	15	20	100	100	0

표 1. 실험 데이터

s1	s2	s3	s4	s5	s6	s7	s8
9100	4500	1400	375	-500	-5000	-7500	0

표 2. 가중치를 곱하여 가공된 데이터 값

표1은 52개의 적중한 데이터 중 한 개를 보여준다. s1 은 가장 왼쪽의 센서, s8은 가장 오른쪽의 센서를 나타내고 각 센서 밑에 있는 숫자는 감지된 조도의 양을 나타낸다. 조도 값은 100으로 정규화 되어있다. PC는 이 데이터를 training set의 데이터와 비교하여 오류가능성을 예측한다. 그런데 라인트레이서는 각 센서의 값을 그대로 이용하지 않고 가중치를 곱하여 가공한 후 사용한다. 따라서 가공되지 않은 데이터를 가지고 training set 데이터와 비교해서는 안 된다. 본 실험에서는 가중치가 곱하여진 변환된 데이터와 training set의 데이터들을 비교하였다. 아래 표3과 같이 training set의 데이터들도 변환하여 사용했다. 표2는 표1에 가중치를 곱하여 가공한 데이터를 보여준다.

s1	s2	s3	s4	s5	s6	s7	s8	유클리디안 거리	클래스	순위
9000	6525	2400	800	-1025	-2850	-6600	-2500	4153.236689619	오류	1
7700	1800	1000	700	-1450	-4450	-6375	-3200	4714.472398901	오류	2
5700	2400	400	325	-875	-2800	-6375	-1300	4990.866658207	오류	3
8200	4350	1600	400	-325	-400	-3000	-5500	8518.435889293	정상	4
9000	5100	2750	525	-600	-3400	-5475	-8800	9291.158431541	정상	5

표 3. k-NN 적용 시 상위5개의 데이터 샘플

PC는 가공된 training set의 데이터와 표3의 데이터를 가지고 유클리디안 거리를 구한다. 이 거리가 짧은순서대로 상위5개를 추출한다. 추출된 데이터는 위의 표3과 같다. 표3을 보면 3개의 데이터는 오류이고 2개의 데이터는 정상이다 따라서 이것은 오류로 판정된다.

아래의 그림은 상위5개의 데이터로써 선택된 센서 데이터 값이 어떻게 분포되어 있는지를 보여준다. 그림7은 오류 클래스의 데이터들과 표1의 데이터를 함께 비교하여 나타낸 그림이다. 분포를 보면 대체적으로 s3과 s4에 그리고 s8에서 적은양의 빛이 검출된다는 것을 알 수 있다. 그러나 그림8의 정상 클래스 데이터 값 분포는 s4, s5에서만 적은양의 빛이 검출된다. 같은 클래스의 데이터들은 서로 같은 분포의 형태를 보였다. 클래스간 분포의 차이는 쉽게 확인할 수 있을 정도로 극명하게 나타났다.

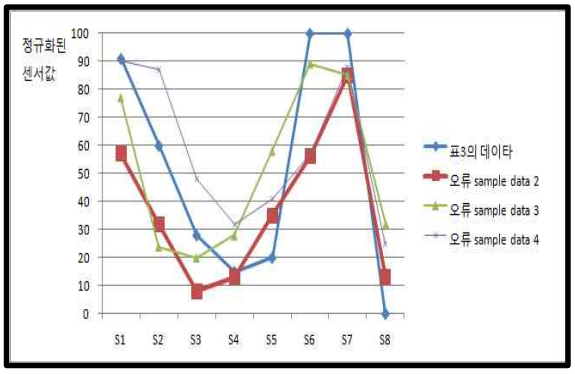

그림 7. 오류 클래스의 센서데이터 값 분포

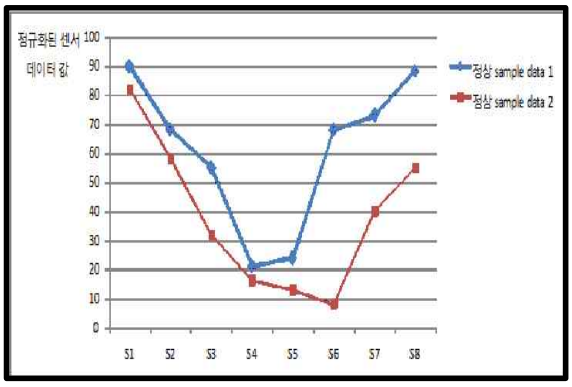

그림 8. 정상 클래스의 센서데이터 값 분포

5. 결론

본 논문에서는 라인트레이서와 PC 간의 블루투스 환경을 구축하여 실시간으로 데이터를 수집하고, 이를 마이닝하여, 미래의 데이터가 오류의 가능성이 있는지의 여부를 판단할 수 있는 시스템을 구현하였다. 이것은 비단 라인트레이서만이 아니라 다양한 임베디드 시스템에 적용될 수 있으리라 생각된다. 실제로 물리환경과 맞닿아 있는 시스템의 경우 일반적인 컴퓨팅 환경처럼 개발자가 적용한 알고리즘대로 수행하지 않는다. 빛의 산란, 통신 흐름의 문제, 센서 의 민감도 등 다양한 외란들이

곳곳에 숨어있다. 그러나 이러한 모든 상황을 고려하여 시스템을 개발 하는 것은 많은 시간과 노력이 요구된다. 또한 점차 실세계와 컴퓨팅이 결합되어 갈수록 알려지지 않은 물리세계의 오류들이 점차 드러날 수 있다. 따라서 이것의 원인을 모두 찾아내고 해결 하는 것 보다는, 그것의 패턴을 분석하고 예측하는 시스템이 더욱 적합할 것이다.

본 논문에서는 이러한 시스템을 빛의 산란에 민감한 라인트레이서를 모델로 하여 소개하였다. 그러나 이 시스템을 원하는 환경에 적용하기 위해서는 두 가지가 필요하다. 우선 모니터링한 데이터가 오류 데이터인지 아닌지를 판단할 수 있는 근거가 있어야 한다. 라인트레이서의 경우에는 모든 센서에 100 이 들어올 경우 라인이탈, 즉 오류라는 근거가 있었다. 이러한 근거는 주어진 시스템 환경에 의존적일 수밖에 없다. 따라서 이것을 모든 시스템에 적용 가능한 일반화된 형태로 만드는 것이 현재로써는 어렵다고 할 수 있다. 두 번째는 어느 선까지 데이터 마이닝을 적용할 것인가 하는 문제이다. 데이터 마이닝은 100%를 보장하지 않는다. 참이 거짓일수 있고 거짓이 참일 수도 있다. 따라서 어디까지 신뢰하고 적용할지는 개발자의 몫이다. 본 논문에서 모델로 제시한 라인트레이서의 경우에는 마이닝된 정보를 근거로 하여 정지 동작을 수행하게 하였다. 그 이상의 제어권을 마이닝된 정보에 맡기는 것은 라인트레이서가 이탈하게 만들 수 있기 때문이다.

소프트웨어의 오류는 프로그램 코드와 컴퓨팅 환경이라는 라는 제한된 환경에서 일어나는 현상이지만, 물리적 오류는 상대적으로 범위가 넓으며, 그만큼 예측이 어렵다. 그러나 본 연구를 통해 이러한 오류도 어느 정도 예측 가능하게 되었다. 향후에는 각 개발 환경에서 어떤 근거를 토대로 데이터를 분류할 것인지, 어느선 까지 데이터 마이닝을 적용할 것인지를 연구 할 예정이다.

감사의 글

본 연구는 서울시 산학연 협력사업(과제번호 CR070019)과 한국연구재단 기초연구사업(2009-0070392)의 지원을 받았습니다.

6. 참고 문헌

[1] HCS12 T-Board Mannual http://elmicro.com/files/elmicro/hcs12tbv100_manual_en.pdf.

[2] Getting started with the Mtorola HCS12 family us-ing Metrowoks CodeWarrior. http://www.discover.uottawa.ca/~qchen/TA/elg4911/an1002_codewarrior_getting_started.pdf.

[3] P. Tan, M. Steinbach and V. Kumar. “Introduction to Data Mining”. Pearson Addison-Wesley, 2006.

[4] M.S. Chen, J. Han and P. Yu, “Data Mining : An Overview from Database Perspective”. IEEE Trans. On Knowledge and Data Engineering, 1997.

[5] Segaran, Toby, “Programming Collective intelligen-ce”, Oreilly & Associates Inc, 2007.

[6] 장봉원, 이동욱, 김재훈, 조위덕, “유비쿼터스 모니터링 시스템에서의 연관분석을 활용한 다중상황 적응적 데이터 감시 전략”, 2008 한국 컴퓨터 종합학 대회 논문집 Vol.35, No.1(B), 394-399, 2008.

[7] 이수용, 김지홍, 김용현, 음화석, 오명근, 홍윤식, “ZigBee 모듈을 이용한 휴대용 인슐린 펌프의 실시간 모니터링 시스템”, 2007 한국컴퓨터 종합학술대회 논문집 Vol.35, No.1(B), 2007.