KIPS Tr. Software and Data Eng.
Vol.4, No.4 pp.151~160 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.4.151

Methods for Enhancing Reliability of On-Ground IoT Applications

Dong Ha Shin† · Seung Ho Han† · Soo Dong Kim†† · Jin Sun Her†††

ABSTRACT

Internet-of-Things(IoT) is the computing environment to provide valuable services by interacting with multiple devices, where diverse devices are connected within the existing Internet infrastructure and acquire context information by sensing. As the concern of IoT has been increased recently, most of the industries develop many IoT devices. And, many people are focused on the IoT application that is utilizing different technologies, which are sensor network, communication technologies, and software engineering. Developing on-ground IoT application is especially even more active in progress depending on increasing of on-ground IoT devices because it is possible for them to access dangerous and inaccessible situation. However, There are a few studies related IoT. Moreover, since on-ground IoT application, which is different from typical software application, has to consider device's characteristics, communication, and surround condition, it reveal challenges, decreasing reliability. Therefore, in this paper, we analyze reliability challenges related to maturity and fault tolerance, one of reliability attributes, occurring in developing on-ground IoT applications and suggest the effective solutions to resolve the challenges. To verify proposed the challenges and solutions, we show result that is applying the solutions to applications. By presenting the case study, we evaluate the effectiveness of applying the solutions to the application.

Keywords : Internet-of-Things, IoT Devices, On-Ground IoT, IoT Application Development, Enhancing Reliability

지상용 IoT 애플리케이션의 신뢰성 향상 기법

신 동 하† · 한 승 호† · 김 수 동†† · 허 진 선†††

요 약

사물 인터넷(Internet-of-Things, IoT)은 무선 인터넷으로 연결된 다양한 디바이스에게서 센싱을 통해 정보를 획득하고 여러 디바이스와 협업을 통해 사용자에게 유용한 서비스를 제공하는 컴퓨팅 환경을 말한다. 최근 사물 인터넷에 대한 관심이 높아짐에 따라 다양한 IoT 디바이스들이 개발되고 있으며, 특히 사람이 접근하기 어렵거나 위험한 환경에서 활용 가능한 지상용 IoT 디바이스에 대한 관심이 높다. 또한 이를 활용한 IoT 애플리케이션은 센서 네트워크나 통신 기술, 다양한 설계 기법 등의 기술이 활용되어 사용자에게 다양하고 유용한 기능을 제공한다. 이에 따라 지상용 IoT 애플리케이션에 대한 연구와 개발이 활발히 진행 중이다. 그러나 사물 인터넷은 최근 소개된 신기술이기 때문에 관련 연구에 대한 자료가 부족하며, 특히 지상용 IoT 애플리케이션은 기존의 전형적인 소프트웨어와는 다르게 디바이스의 특성이나 통신 환경, 주변 환경 등으로 인한 애플리케이션의 신뢰성을 저해하는 이슈들이 발생할 수 있다. 본 논문에서는 지상용 IoT 애플리케이션의 신뢰성 중 성숙도와 오류 관리성 관련 이슈들을 분석하고, 이를 효과적으로 해결할 수 있는 신뢰성 향상 기법들을 제시한다. 그리고 지상용 IoT 디바이스 중 Sphero Ball을 활용한 애플리케이션 개발을 통해 본 논문에서 제시한 이슈들을 도출하고 이를 해결하기 위한 솔루션을 적용 및 활용한 사례를 보여줌으로써 연구의 실효성을 보여준다.

키워드 : 사물 인터넷, IoT 디바이스, 지상용 IoT 디바이스, IoT 애플리케이션 개발, 신뢰성 향상

1. 서 론

Internet of Things(IoT)는 사물 혹은 디바이스가 무선 네트워크에 연결되어 이를 기반으로 다양한 서비스를 제공할 수 있는 기술이다. IoT 애플리케이션은 무선 네트워크를 통한 IoT 디바이스의 센싱과 액추에이팅을 함으로써 사용자에게 유용한 서비스를 제공한다[1].

다양한 IoT 디바이스 중, 전쟁터 혹은 지하와 같은 사람이 접근하기 어렵거나 위험한 환경에서 유용한 지상용 IoT 디바이스가 널리 사용되고 있다. 지상용 IoT 디바이스는 지상에서 바퀴나 디바이스 자체의 모터 회전을 통해 자체 이동성을 제공할 수 있는 디바이스로, 이를 활용한 지상용 IoT

※ 이 논문(저서)은 2011년도 정부재원(교육부)으로 한국연구재단의 지원을 받아 연구되었음(NRF-2011-35C-B00109).
※ 이 논문은 2014년도 한국정보처리학회 추계학술발표대회에서 'Sphero Ball 사물인터넷 애플리케이션 개발의 기술적 이슈'의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 숭실대학교 컴퓨터학부 학사과정
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
††† 정 회 원 : (주) 스마티랩 연구원
Manuscript Received : January 14, 2015
First Revision : March 17, 2015
Accepted : March 19, 2015
* Corresponding Author : Jin Sun Her(annie8js@gmail.com)

애플리케이션 개발이 다양하게 이루어지고 있다.

지상용 IoT 애플리케이션은 기존의 전형적인 소프트웨어와 다르게 지상에서 이동 가능한 디바이스를 사용하므로 디바이스의 특성이나 주변 환경에 따라 서비스 제공이 원활하게 이루어지지 않을 수 있다. 즉, 애플리케이션의 신뢰성을 저해시킬 수 있는 요인이 많아 품질이 떨어질 수 있다.

IoT 애플리케이션은 일반적인 소프트웨어 외에도 주변 환경, 디바이스의 특징, 그리고 네트워크까지 고려하여 개발해야 신뢰성이 높아진다. 그러나 현재까지 지상용 IoT 디바이스의 특성과 주변 환경으로 인해 애플리케이션의 품질을 저해할 수 있는 이슈에 대한 연구가 충분하지 않다. 즉, 지상용 IoT 애플리케이션의 신뢰성을 향상시킬 수 있는 기법이 부족하여 애플리케이션 개발의 어려움과 운용 시 오류가 발생한다.

그러므로 본 논문에서 지상용 IoT 애플리케이션 개발 시 신뢰성을 저해할 수 있는 여러 이슈들을 나열하고, 이에 대한 향상 기법들을 제시한다. ISO 9126 표준에서는 신뢰성을 성숙도, 오류 관리성, 복구성 세 가지로 분류한다[2]. 그중 디바이스의 고장을 복구시키는 것은 소프트웨어로 처리하기 어려우므로 성숙도와 오류 관리성에 대한 향상 기법을 제시한다. 지상용 IoT 애플리케이션 신뢰성은 대부분 이동 관련 기능에서 나오기 때문에 정확한 위치를 측정하고 특정 지점으로 이동하는 기능에서 도출된다. 그리고 이동 기능을 저해하는 충돌 및 주변 환경과 관련된 이슈들을 도출하고, 각 이슈에 대한 신뢰성을 향상시킬 수 있는 기법들을 제시한다.

본 논문의 2절에서는 관련 연구를 조사하고, 3절에서 디바이스의 이동과 표면 환경에 관련된 성숙도 이슈 및 향상 기법을 제시한다. 그리고 4절에서는 디바이스의 충돌 감지와 네트워크 관련 오류 관리성 이슈 및 향상 기법을 제시한다. 마지막 5절에서는 제시된 성숙도와 오류 관리성 향상 기법을 적용하여 본 연구의 효용성을 검증하기 위해 지상용 디바이스인 Sphero Ball[3] 이동 기반 애플리케이션을 개발하는 사례연구를 수행한 결과를 보여준다. 본 논문에서 제시된 신뢰성 향상 기법을 숙지하고 지상용 IoT 애플리케이션 개발 시 적용하면 애플리케이션의 신뢰성을 높이고, 운용 시 오류를 예방 및 해결하여 높은 품질을 기대할 수 있다.

2. 관련 연구

이전 연구에서는 지상용 IoT 디바이스 중 Sphero Ball을 제어하는 애플리케이션 개발 시 발생하는 기술적 이슈를 다루고 있다. 그러나 이전 관련 연구에서는 여러 지상용 IoT 디바이스 중 Sphero Ball에만 의존적이며, 기술적 이슈에 대한 솔루션이 제시되지 않았다. 이에 따라 본 논문에서는 Sphero Ball뿐만 아니라 다양한 지상용 IoT 디바이스를 다루고 있으며, 지상용 IoT 디바이스 애플리케이션 개발 시 발생할 수 있는 기술적 이슈를 자세히 다루고 있다. 또한, 발생하는 기술적 이슈에 대한 솔루션을 제시하기 위해 [4]의 연구를 확장하여 사례연구를 통해 적용 사례를 보여줌으로써 연구의 실효성을 높이고 있다.

Miorandi의 연구는 IoT 개념을 구현할 때 시스템 수준에서 디바이스에 대한 이질성, 확장성, 효율성, 그리고 보안 등의 관리를 나열하고 이와 관련된 기술적 이슈를 나열하였다[5]. 이 연구에서 제기된 이슈는 디바이스 식별 관련 이슈, 분산된 디바이스 관리 관련 이슈, 분산된 데이터 처리 관련 이슈, 보안 관련 이슈 등이다. 이 연구는 IoT 애플리케이션 개발에 관련된 이슈들 위주로 언급하고, 기존 연구 기법들이 어느 정도 해결할 수 있는지를 기술하였지만, IoT 디바이스의 특성을 고려한 솔루션을 제시할 필요가 있다.

La의 연구는 IoT 애플리케이션 개발 시 일반적인 소프트웨어 시스템과는 다른 비전형적인 이슈들을 나열하고 이에 대한 해결 방법을 제시하고 있다[6]. 이 연구는 다양한 디바이스로 인한 이질성, 자체 이동성, 그리고 물리적인 제약성으로 구분하여 이에 대한 솔루션과 설계의 기반을 제시하였다. 그러나 자체 이동성으로 인한 네트워크 연결의 불안정에 대해서만 언급하고 다른 특성은 고려하지 않는다.

Marsman의 연구는 사용자와 IoT 디바이스 관리 운용 시스템이 제어 권한을 공유하여 Sphero Ball의 실시간 제어 비용을 최소화하는 기법을 제안한다[7]. 이 연구는 IoT 디바이스에 대한 명령 주체의 변환을 지원하며, 각 주체에서 서로 다른 방향으로 IoT 디바이스 이동 명령이 발생하였을 때 처리 기법을 제시하고 있다. 그러나 디바이스의 특성 때문에 발생할 수 있는 다른 이슈에 대한 방안이 필요하다.

현재까지 IoT 관련 연구에서 디바이스의 위치 측정에 대한 중요성이 강조되고 있고, Error Propagation Aware을 이용한 오차 인지[8], 디바이스 간의 거리 측정[9] 등 다양한 기법들이 제시되었다. 제시된 기법들을 활용하면 실내에서 IoT 디바이스의 위치 파악이 가능하다. 그러나 디바이스의 하드웨어적인 특성에 따라 제시된 기법의 적용이 제한되어 IoT 디바이스의 이동거리 및 방향을 고려한 위치 측정 기법이 요구된다.

Chen의 연구는 IoT 애플리케이션이 수행하는 전반적인 기능에 대해서 설명하고 이때 발생할 수 있는 이슈를 제시한다[10]. 제시된 이슈에는 디바이스에 관련된 설치 및 관리, 제한된 특징, 연결 관련, 보안 이슈, 데이터 처리 등이 있다. 또한, 해당 이슈를 해결하기 위한 기법들을 언급하였다. 이 연구는 IoT 애플리케이션 개발 시 이슈들에 대한 솔루션의 방향만을 제시하고 있어 구체적인 솔루션이 필요하다.

본 논문에서는 다양한 지상용 IoT 디바이스를 이용한 애플리케이션 개발 시 신뢰성을 저해시킬 수 있는 이슈들을 도출하고, 이를 해결하기 위한 설계 기법을 제시한다.

3. 성숙도(Maturity) 향상 기법

지상용 IoT 디바이스는 대부분 이동 기반의 기능을 수행하며 사용자에게 원격 제어 서비스를 제공한다. 일반적인 소프트웨어 시스템과 다르게 IoT 애플리케이션 개발은 디바이스를 사용하므로 안정적인 디바이스 제어를 위해 오류에 대한 고장을 피할 수 있는 성숙도를 향상시켜야 한다.

지상용 IoT 애플리케이션은 이동 기반의 기능을 수반하고, 디바이스의 특성상 표면 환경에 제약을 많이 받기 때문에 본 절에서는 지상용 IoT 애플리케이션 개발 시 이동 기반의 기능 관련 성숙도와 표면 환경 제약 관련 성숙도를 저해할 수 있는 이슈와 성숙도를 향상시킬 수 있는 기법을 제시한다.

3.1 디바이스 이동 관련 성숙도 향상 기법

1) 디바이스 이동 관련 이슈

지상용 IoT 애플리케이션 개발에서 이동 기반의 주 기능은 자체 이동성 제공이다. 자체 이동 시 특정 지점까지 이동하는 기능뿐만 아니라 정확한 위치로 찾아가는 기법이 필요하다. 이를 위해서 다음과 같은 두 가지 이슈에 대한 성숙도를 고려해야 한다.

▪ 디바이스 위치 이슈 : IoT 디바이스의 위치 정보는 GPS와 같은 위치 센서를 이용하는 직접적인 위치 측정이 있고, 위치 센서를 탑재하지 못하는 디바이스의 경우 디바이스의 움직임을 계산해 좌표 정보를 제공하는 방법 같은 간접적인 위치 측정이 있다. 예를 들어, 지상용 IoT 디바이스 중 Sphero Ball은 위치 센서가 탑재되어있지 않아 모터의 회전량을 계산하여 간접적인 위치를 측정하여 좌표 정보를 제공한다. 그러나 측정된 정보에 대한 판단이 잘못될 수 있다. Fig. 1은 Sphero Ball이 측정한 위치 좌표와 측정된 위치 정보의 판단 오류로 인한 실제 위치의 차이를 보여준다.

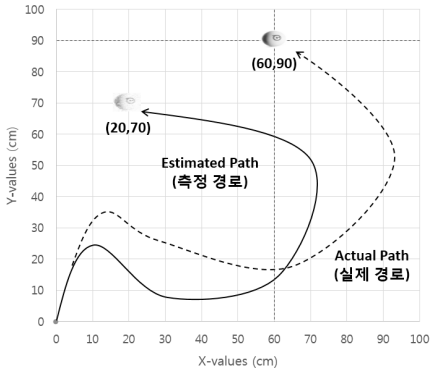


Fig. 1. Difference between Estimated and Actual Path

Sphero Ball과 같이 GPS 등의 위치 측정 센서의 부재로 간접적인 측정을 하는 지상용 디바이스의 경우, 속도나 방향 전환에 따라 공회전이 발생해 실제 이동하는 위치와 측정되는 위치가 달라져 오류가 발생할 수 있다. 예를 들어, Fig. 1의 실선인 측정 경로와 점선인 실제 경로가 달라도 측정되는 모터의 회전량이 같으면 실제 위치 (60, 90)이 아닌 측정된 좌표 위치 (20, 70)으로 간주하게 된다. 오차가 발생한 좌표 정보는 원하는 지점으로 가지 못하는 오류를 발생시키며 위치 기반의 애플리케이션을 만드는 데 성숙도를 저해시킨다.

▪ 디바이스 특정 지점 이동 이슈 : 특정 지점을 정확하게 찾아가는 위치 정보를 제공해주는 센서나 API는 디바이스에 따라 다르다. 만약 어떤 디바이스가 특정 위치로 이동하는 기능에 대한 API 자체를 제공한다면 상관없지만, 단순히 현재 위치 정보만을 제공하는 API나 센서만 있다면 개발자는 특정 위치로 이동하기 위한 방향, 거리 등을 계산해야 되는 문제가 있다.

디바이스의 정확한 위치 도달은 디바이스 현재 속도에 따른 관성에 의한 제동 시 오차가 발생하거나, 방향 전환 시 발생하는 관성 때문에 어려울 수 있다[11]. 대부분의 지상용 IoT 디바이스는 일정 수준의 속도로 이동하며, 이동 중에는 방향 전환이 필요하다. 만약 사용자가 목표하는 위치에 디바이스가 정확하게 도달하길 원하면, 방향 전환 시 관성에 의한 오차와 위치 도달 후 제동 시 관성에 의한 오차까지 고려해야 하는 문제가 있다.

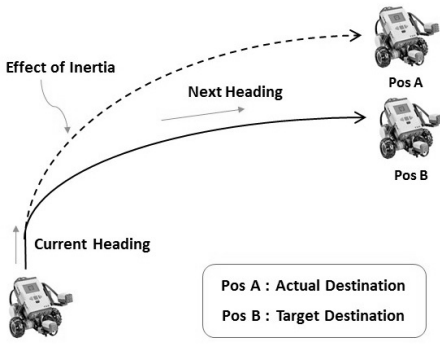


Fig. 2. NXT Moving Error Caused of Inertia

Fig. 2는 지상 이동형 IoT 디바이스 중 하나인 Mindstorm NXT[12]가 방향 전환 시 발생하는 관성에 의해 실제 도달 지점과 목표 도달 지점의 오차를 보여준다.

그림과 같이, Mindstorm NXT는 Pos B인 목표 도달 지점으로 가려고 했지만, 현재 진행 방향에서 방향 전환 시 발생하는 관성에 의해 Pos A인 실제 도달 지점으로 가게 되고, 도착 지점의 오차가 발생한다.

지상용 IoT 디바이스는 디바이스가 이동하는 표면의 재질이나 기울기 등에 따라 이동 시 속도나 방향 등에 영향을 받으며, 특히 바퀴를 사용하는 디바이스보다 Sphero Ball과 같이 표면과 접촉하는 면적이 좁을수록 더 많은 영향을 받는다. 이에 따라 특정 지점으로의 이동 시, 정확도가 많이 감소할 수 있고, 사용자 제어 시 오류가 발생할 수 있다.

2) 디바이스 이동 관련 성숙도 향상 기법

디바이스의 이동 기능은 단순히 특정 지점으로의 이동뿐만 아니라 정확한 위치로 이동해야 한다. 지상용 IoT 디바이스가 해당 기능에 대한 API를 제공하면 상관없지만, 만약 디바이스의 위치만을 제공하는 API나 센서만 존재하면, 개발자는 현재 위치와 특정 지점 사이의 거리와 방향을 구해야 한다. 또한 목표 지점에 도달하여 정지할 경우, 정지 시 발생하는 관성에 의한 오차나 방향 전환 시 발생하는 오차를 계산함으로써 위치의 정확도를 높여야 한다. 즉, 정확한 특정 지점 이동 시 다음과 같은 두 가지를 고려해야 한다.

▪ 특정(목표) 지점과의 거리와 방향 : 현재 위치를 기준으로 특점 지점까지의 거리와 방향은 제공되는 위치 정보가 API인지 센서인지에 따라 계산 방법이 달라질 수 있다. 먼저, 디바이스 자체에서 현재 위치 정보를 API로써 2차원 좌표값을 제공한다면, 다음과 같이 Equation (1)을 통해 거리와 방향을 구할 수 있다.

$$bearing = (\frac{\pi}{2}) - atan2(y_d - y_c, x_d - x_c) \quad (1)$$
$$distance = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$$
x_c: x coordinate of current x_d: x coordinate of destination
y_c: y coordinate of current y_d: y coordinate of destination

Distance and Direction for 2-Dimension

위 식을 통하여, 현재 디바이스로부터 목표 지점까지의 거리와 방향을 구할 수 있다. 만약 디바이스 자체의 위치 센서나 외부 위치 센서를 통해 GPS값을 가져온다면 다음과 같이 Equation (2)와 Equation (3)을 통해 거리와 방향을 구할 수 있다.

$$bearing = atan2(\sin\Delta\lambda \times \cos\phi_d \times \cos\phi_c \times \sin\phi_d - \sin\phi_c \times \cos\phi_d \times \cos\Delta\lambda) \quad (2)$$
ϕ_c: current latitude ϕ_d: destination latitude
λ_c: current longitude λ_d: destination longitude

Direction Based on GPS value

$$a = \sin^2(\Delta\phi/2) + \cos\phi_c \times \cos\phi_d \times \sin^2(\Delta\lambda/2) \quad (3)$$
$$c = 2 \times atan2(\sqrt{a}, \sqrt{(1-a)})$$
$$distance = R \times c$$
R: earth's radius(6,361km) ϕ: latitude λ: longitude
ϕ_c: current latitude ϕ_d: destination latitude
λ_c: current longitude λ_d: destination longitude

Distance Based on GPS Value

위와 같은 식을 이용하여 GPS 좌표 기반의 디바이스로부터 목표 지점까지의 거리와 방향을 구할 수 있다.

▪ 관성에 의한 오차 : 특정 지점으로의 거리와 방향을 구한다고 해도 정확하게 해당 지점으로 이동한다는 보장은 없다. 특히 이동 중 방향 전환이나 내리막길 등의 다양한 상황에 따라 오차가 발생할 수 있다. 이런 경우 오차가 생기는 가장 큰 원인은 관성에 의한 오차이다. 관성이 발생하는 상황은 크게 디바이스 이동 중 방향 전환 시와 디바이스가 정지 시이다.

먼저, 방향 전환 시 발생하는 관성에 의한 오차는 목표 지점까지의 남은 거리를 지속적으로 계산함으로써 막을 수 있다. 만약 현재 남은 거리가 이전 남은 거리에 비해 증가할 경우에는 방향 전환 등의 문제로 오차가 발생한 경우이므로 이 경우에 다시 방향을 계산하고 진행한다. 이에 대한 알고리즘은 다음과 같다.

```
Begin
while(remainingDistance > 10) { // 10cm
if(currentRemainingDistance
                    > preRemainingDistance) {
    // recalculate bearing and distance
    bearing = getBearing(curX, curY, tarX, tarY);
  }
  drive(bearing, deviceSpeed);
}
End
```

또한 디바이스가 정지 시 발생하는 오차는 디바이스의 현재 좌표와 속도를 계산하고, 목표 지점까지의 남은 거리에 따라 속도를 줄임으로써 오차를 최소화시킨다. 즉, 남은 거리가 일정값 이하가 되는 순간부터, 남은 거리가 감소함에 따라 일정 속도만큼을 감소시킴으로써 정지 시의 오차를 줄인다. 다음 알고리즘은 목표 지점과의 남은 거리가 40% 이하가 되었을 때 거리가 10% 감소함에 따라 디바이스 속도를 20%씩 감소시키는 알고리즘의 예이다.

```
Begin
var criterionDistance
            = getDistance(curX, curY, tarX, tarY);
var criterion = criterionDistance * 0.1;
criterionDistance *= 0.4;
while(remainingDistance > 10) { // 10cm
  if(currentRemainingDistance < criterionDistance) {
    criterionDistance -= criterion;
    deviceSpeed *= 0.2;
  }
  drive(bearing, deviceSpeed);
}
End
```

이와 같이, 특점 지점으로 이동 시, 먼저 방향과 거리를 계산 후 이동을 수행한다. 이동 시에는 지속적으로 남은 거리를 계산하여 잘못된 방향으로 이동 시, 방향을 다시 계산함으로써 오차 발생을 최소화시킨다. 그리고 남은 거리가 감소함에 따라 디바이스 속도값을 줄임으로써 정확한 위치 이동과 관련된 기능의 성숙도를 향상시킬 수 있다.

3.2. 실행 표면 환경 관련 성숙도 향상 기법

1) 실행 표면 환경 관련 이슈

지상용 IoT 디바이스 운영 시 신뢰성 있는 디바이스 제어 환경을 제공하려면, 디바이스 제어에 직접적인 영향을 주는 요소들에 대한 고려가 필요하다. 특히, 지상용 디바이스는 표면과 직접적인 접촉이 있기 때문에 표면의 마찰력이

나 경사도, 낙차 등의 표면 환경 영향을 받는다. 표면의 마찰력은 디바이스가 이동하는 표면 재질과 마찰 계수에 따라 달라질 수 있으며, 디바이스 자체의 스펙(Spec)에 따라서도 달라질 수 있다. 표면의 경사도는 디바이스가 이동하는 경사도를 말하며, 낙차는 계단과 같이 급격한 낙차 상황을 말한다. 이와 같은 표면 환경 요소를 고려하지 않으면, 디바이스 제어의 어려움이나 디바이스 손상 등의 문제가 발생할 수 있다.

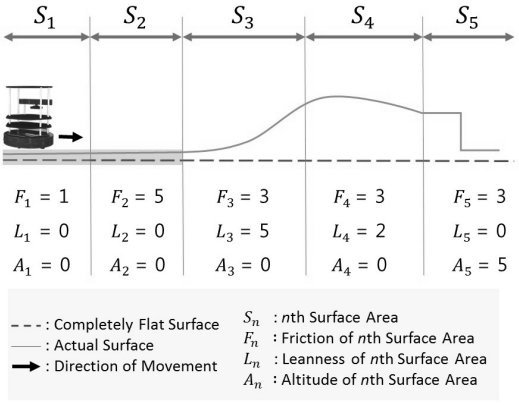


Fig. 3. Various Surface Environment

Fig. 3은 지상 이동형 IoT 디바이스 중 하나인 터틀봇(Turtlebot)[13]의 이동 시 다양한 표면 환경을 보여준다. 그림에서는 세 가지의 표면 특성을 고려하고 있으며 각각 마찰 정도, 경사 정도, 낙차 정도를 나타내고 각 특성의 값은 상대적인 값으로 0~5의 범위를 갖는다.

위 그림과 같이, 터틀봇이 각 표면에서 동일한 속도로 동작하도록 명령 시 상대적 수치로 표현된 환경 특성에 따라 마찰력에 영향을 받는 S1에서는 기존 속력보다 빠르게, S2에서는 느리게 이동하게 된다. 경사가 있는 S3와 S4에서는 경사에 각도에 따라서 속력의 변화가 발생할 수 있다. 또한, 디바이스의 특성상 낙차가 존재하는 S5에서는 주행이 제한적이다.

2) 실행 표면 환경 관련 성숙도 향상 기법

경사도나 낙차의 경우, 디바이스마다 하드웨어적 한계가 존재하지만, 표면의 마찰력에 따른 오차는 간접적인 위치 측정을 하는 경우에 영향을 받으며, GPS 같은 직접적인 위치 측정 좌표는 문제가 되지 않는다. 간접적인 위치 측정의 경우에는 마찰력이 낮은 경우에 바퀴의 공회전 등이 발생하여 좌표값에 오류가 생길 수 있기 때문이다. 이에 따라 다음과 같은 두 단계를 통한 해결 방법을 제시한다.

첫 번째 단계에서는 초기 디바이스 연결 시, 자동으로 일정 시간 동안 calibrate를 수행한다. calibrate 수행 동안 디바이스의 속도 변화량을 계산하고 좌표를 얻어, 속도에 따른 좌표 변화량을 계산한다. 또한 이때의 마찰력을 기준 마찰력으로 설정한다.

두 번째 단계에서는 디바이스 이동에 따라 주기적으로 속도를 계산하고 좌표값을 얻어, 속도에 따른 예상 좌표를 계산한다. 이때, 계산된 예상 좌표와 현재 좌표가 다를 경우, 초기 설정한 기준 마찰력에 대해 마찰력이 변화된 것으로 판단하여 이에 대해 다음과 같이 적용한다. 각 좌표에 따른 거리를 Before(X_1, Y_1), Expected(X, Y), Current(X_2, Y_2)라고 정의하고, Before < Expected라고 가정하며 절댓값으로 비교한다.

S_n(Current ≥ Expected) : 마찰력 F_n의 값이 커진 것으로 판단하고, 그대로 진행한다.

S_n(Current < Expected) : 마찰력 F_n의 값이 작아진 것으로 판단하고, 현재 좌표값을 예상 좌표값으로 바꾼다.

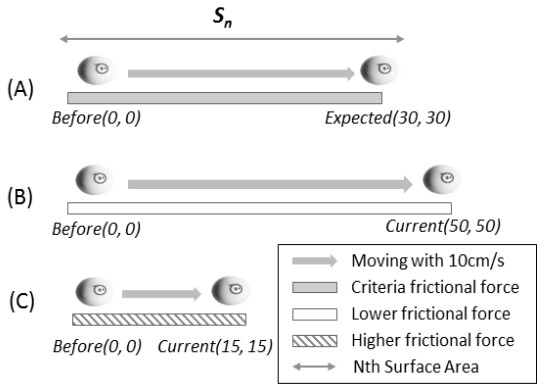


Fig. 4. Compare to Coordinates with Frictional Force

Fig. 4는 어떤 구간 S_n에서 마찰력에 따른 디바이스의 좌표 변화를 비교하고 있다. 그림의 (A)는 사용자가 10cm/sec의 속도를 입력하고 3초 후의 예상 좌표이다. (A)의 표면은 calibrate 수행 후 기준이 되는 마찰력을 가지고 있어 예측되는 좌표 결과가 나온다. 이후 디바이스 이동 시 동일한 시간 간격으로 좌표 변화를 분석하게 된다. 이에 따라 기준 마찰력보다 마찰력이 작아진 경우에는 (B)와 같이 공회전으로 인한 잘못된 좌표값이 할당되게 되고 이 경우 현재 좌표가 아닌 예상 좌표값을 넣어 잘못된 좌표값을 보정한다. 만약 기준 마찰력보다 마찰력이 커진 경우에는 같은 속도와 시간 동안 이동한 거리는 작아지지만 실제 좌표값은 잘못되지 않았으므로 계속 수행한다.

표면의 경사도는 경사도가 높은 경우와 낮은 경우로 나뉘며 경사도가 낮은 경우, 즉 내리막의 경우에는 속도에 따른 오차가 생기는 문제이므로 3.1절에서 제시한 기법을 적용한다. 경사도가 높은 경우, 즉 오르막의 경우에는 예상 좌표가 이전 좌표보다 작거나 같은 경우로서, 이 경우에 빗면에 가해지는 힘보다 작은 속도값을 주게 되면 디바이스가 역행할 위험이 있다. 이에 따라 디바이스가 역행하지 않도록 자체적으로 역행 시에 디바이스 속력을 올림으로써 해결 가능하다. 즉 현재 좌표가 이전 좌표보다 작아지는 경우가 발생하지 않을 때까지 속도를 올리게 된다. 이를 통해, 환경과 관련된 오류를 예방하여 애플리케이션의 높은 성숙도를 기대할 수 있다.

4. 오류 관리성(Fault Tolerance) 향상 기법

IoT 디바이스는 자체 이동성을 가지고 있고 사용자와 네트워크를 통해 커뮤니케이션 한다. 이러한 IoT 디바이스의 특징 때문에 애플리케이션 수행 시 오류가 발생하면 서비스를 제공받기 어렵다. 특히, 지상용 IoT 디바이스는 비행용 IoT 디바이스와 다르게 실내에서 많이 이용되기 때문에, 건물의 벽과 같은 장애물과 충돌 시 발생할 수 있는 오류에도 일정 수준의 서비스를 제공할 수 있어야 한다. 또한, 이런 특징 때문에 발생할 수 있는 네트워크 오류에 대한 관리도 필요하다. 따라서 본 절에서는 지상용 IoT 애플리케이션 개발 시 충돌 감지 기능과 네트워크 관련 이슈를 설명하고 오류 관리성을 향상시킬 수 있는 기법을 제시한다.

4.1 충돌 감지 관련 오류 관리성 향상 기법

1) 충돌 감지 관련 이슈

지상용 IoT 디바이스는 실내에서 많이 이용되기 때문에, 장애물을 감지하고 충돌 상황을 예측하거나 충돌이 발생할 수 있는 예외 상황을 제거하는 것이 바람직하다. 지상용 디바이스에 근접 센서가 탑재되어있으면 장애물을 미리 감지하고 충돌이 발생하지 않도록 처리할 수 있지만, 디바이스 특성상 근접 센서가 탑재될 수 없는 경우 이동 중 발생하는 장애물과의 충돌은 예측할 수 없기 때문에 충돌이 발생할 경우 감지하여 처리하는 기능이 더욱 중요하게 다뤄진다. 예를 들면, Sphero Ball은 공 모양의 디바이스 특성상 근접 센서를 탑재할 수 없기 때문에 충돌을 예측할 수 없다. 이에 따라 충돌을 정확히 감지하고 이를 신뢰성 있게 처리할 수 있어야 한다.

근접 센서를 탑재하지 않는 디바이스 중 일부는 충돌을 감지할 수 있는 기능을 제공한다. 그러나 이 기능은 충돌이 아닌 상황에서 충돌로 감지하는 상황도 발생할 수 있다. 예를 들면, Sphero Ball의 충돌 감지는 회전하는 내부 축의 충격값을 활용하여 사용자가 설정한 임계값을 초과하는 경우 충돌 상황으로 처리한다. 이때, 충돌이 아닌 상황에서 임계값을 초과하는 경우 문제가 발생한다. Fig. 5는 Sphero Ball이 이동 중 충돌로 감지할 수 있는 상황을 보여준다.

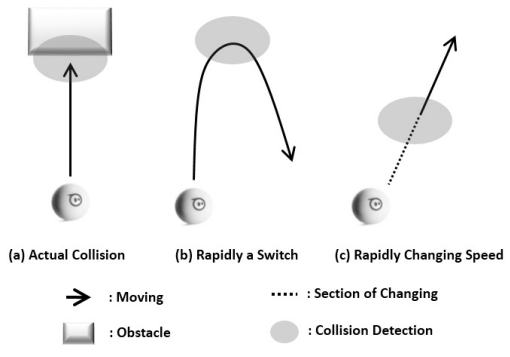


Fig. 5. An Example for Situation of Collision Detection

Fig. 5에서는 Sphero Ball이 실제로 장애물과 충돌하여 충돌 상황으로 감지하는 정상적인 작동 (a)와 급격한 방향 전환 (b), 빠른 속도 변화 (c)로 내부의 충격값이 설정된 임계값을 초과해 충돌로 감지하는 상황을 보여준다. (b)와 (c)의 경우 실제로 충돌하지 않았지만, 오판단 때문에 충돌로 감지하여 문제가 발생한다. 즉, 디바이스 구동 중 발생할 수 있는 비정상적인 충돌 감지는 처리에 어려움이 있다.

오판단으로 인한 비정상적인 충돌 감지는 지상용 IoT 애플리케이션 실행 시 디바이스 이동의 신뢰성을 낮추고 기능에 제한을 준다.

2) 충돌 관련 오류 관리성 향상 기법

지상용 IoT 디바이스의 정확한 충돌 감지를 위해 실제 충돌 시 변화량과 오판단으로 인한 변화량의 패턴을 비교 및 분석하여 가장 신뢰 있는 임계값을 설정해야 한다. 즉, 실제 충돌 상황의 패턴은 방향 전환이나 빠른 속도의 변화로 인한 변화량과 다르기 때문에 이를 패턴으로 비교하는 것이다. 이때, 정면에서뿐만 아니라 측면에서의 충격값을 고려하여 패턴을 비교해야 한다. 이를 위해 다음과 같은 3단계를 통해 디바이스 충돌 감지 및 처리 방법을 제시한다.

첫 번째 단계는 디바이스의 실제 충돌을 감지하기 위해서 충돌 시 변화하는 내부 충격량을 측정할 필요가 있다. 각도에 따라 충돌이 발생했을 때의 값을 추출하도록 한다.

두 번째 단계는 충돌로 오판단하게 되는 상황을 구분한다. 예를 들면, 급격한 속도의 변화 혹은 방향 전환으로 인해 축의 변화량이 커져 충돌이라고 판단하게 되는 경우가 해당된다. 구분된 상황에서 충돌이라고 감지하게 되는 값을 추출한다.

세 번째 단계는 실제 충돌과 오판단으로 인한 충돌 상황을 비교하여 실제 충돌과 패턴 매칭이 되면 충돌로 감지한다. 제시된 3단계를 통하여 신뢰성 있는 충돌 감지를 한 후, 충돌이 일어나기 전 위치로 돌아가는 등의 처리를 해주면 충돌 때문에 발생할 수 있는 오류에도 일정 수준의 신뢰성 있는 서비스를 제공할 수 있다.

4.2 네트워크 관련 오류 관리성 향상 기법

1) 네트워크 관련 이슈

여러 지상용 IoT 디바이스는 사용자에게 원격 디바이스 제어 기능을 제공하기 때문에, 무선 통신 환경은 필수적이다. 이에 따라 대부분의 지상용 IoT 디바이스는 다양한 통신 환경을 제공하며, 그중 가장 많이 사용되는 통신 방법은 블루투스와 Wi-Fi이다.

블루투스 통신은 근거리 무선 통신망으로서, 블루투스가 지원되는 스마트 기기와 IoT 디바이스가 페어링(Pairing)을 통해 연결하게 된다. 이 후, 모터 제어나 디바이스 제어, 센서값, 상태 정보, 동영상, 사진 등의 여러 데이터의 교환을 위해 지속적으로 패킷을 주고받게 된다. Wi-Fi 통신 또한 블루투스 통신과 유사하다. 다만, Wi-Fi의 경우 통신 가능

한 범위가 더 넓은 대신, 비용이나 전력 소비 등이 많이 소모되기 때문에 디바이스에 따라 적당한 프토로콜 방식을 사용하게 된다. 대표적인 Wi-Fi를 사용하는 지상용 IoT 디바이스로는 터틀봇, 점핑 스모[14] 등이 있다.

지상용 IoT 디바이스는 이와 같이 무선 통신 환경이 필수적이지만, 디바이스의 자체 이동성 때문에 다음과 같은 문제가 발생할 수 있다.

- 잦은 신호 강도 변화로 인한 네트워크 불안정
- 장애물로 인한 네트워크 불안정
- 네트워크의 통신 범위 한계

사용자에게 신뢰성 있는 디바이스 제어를 하기 위해서는 통신 환경이 안정적이어야 한다. 그러나 대부분의 지상용 IoT 디바이스는 자체 이동성을 갖기 때문에, 통신 거리의 변화가 심하고, 이에 따라 통신 신호 강도 또한 변화가 심하게 된다. 이러한 신호 강도의 변화는 불안정한 네트워크 환경을 만들게 되며, 통신의 지연이나 끊김 등의 문제가 발생할 수 있다. 또한 네트워크에서 제공되는 통신 범위의 한계 때문에 끊김이나, 벽이나 장애물 등으로 인한 네트워크 불안정 등의 문제가 발생할 수 있다.

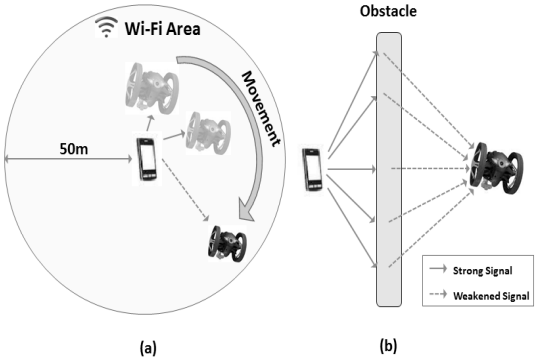


Fig. 6. An Unstable Network Connection

Fig. 6는 Wi-Fi를 사용하는 점핑 스모의 불안정한 네트워크 연결 상태 모습을 보여준다. (a)의 경우, 점핑 스모가 제공되는 네트워크 범위 내에서 이동하는 모습이다. 그림에서와 같이, 제어하는 스마트 기기로부터 멀어질수록 Wi-Fi의 신호가 약해지게 된다. (b)의 경우, 제어하는 스마트 기기와 점핑 스모 사이에 장애물이 있어 신호가 약해지는 모습을 보여준다. 이에 따라 네트워크가 불안정해지고, 끊어지는 문제가 발생할 수 있으며 디바이스 제어가 어려워져 사용자에게 신뢰성 있는 기능을 제공하지 못한다.

2) 네트워크 관련 오류 관리성 상상 기법

지상용 IoT 디바이스 제어에서 네트워크 관련 문제는 지속적인 네트워크 모니터링(Monitoring)을 통해서 해결할 수 있다.

모니터링은 애플리케이션과 디바이스가 연결된 시점에 자동으로 수행되며, 애플리케이션이 종료될 때까지 지속적으로 디바이스와의 연결을 체크하게 된다. 모니터링을 하는 주기는 자주 할수록 연결이 끊겼을 때 보다 빠르게 알 수 있지만, 모니터링을 수행하고 있는 기기의 자원을 고려하였을 때, 다음과 같은 기준으로 주기를 고려해야 보다 효율적일 수 있다.

- 신호 강도가 일정 수준보다 낮은 경우
- 신호 강도의 변화가 심한 경우
- 네트워크 기반의 여러 기능이 동시 수행될 경우

위와 같이, 거리나 장애물에 따라 신호 강도가 낮거나 변화가 심한 경우에는 네트워크가 불안정하여 네트워크가 끊길 확률이 높으므로 주기를 보다 높게 설정하여 자주 모니터링을 하게 한다. 네트워크 모니터링에 대한 알고리즘은 다음과 같다.

```
Begin
var period = 10 // initial value
var isConntected = TRUE
while(end of application) {
  period = checkNetCondition()
  isConneted = checkConnection()
  if(isConnected == FALSE) {
    reconnect()
    continue
  }
  sleep(period)
}
End
```

위 알고리즘과 같이, 초기의 주기를 설정하고 디바이스와 연결이 되면, 현재의 네트워크 상태와 연결 상태를 체크하는 모니터링이 수행된다. 모니터링 과정에서 매 주기마다 네트워크 상태를 체크하여, 주기를 조정하게 되며, 연결이 끊겼을 시에는 재연결을 수행하게 된다.

5. 사례 연구

본 논문에서 제시된 설계 기법을 검증하기 위해 지상용 디바이스 중 Sphero Ball을 이용하여 안드로이드용 IoT 애플리케이션을 개발에 적용하였다.

5.1 사례연구 도메인

Sphero Ball IoT 애플리케이션은 기본적으로 Sphero Ball의 자체 이동성을 제공하는 오토 모드를 제공한다. Fig. 7은 애플리케이션의 오토 모드를 보여준다. 화면의 START 버

튼을 통해 애플리케이션의 자동 움직임을 볼 수 있고, 일정 시간이 지나면 출발 지점으로 돌아오는 기능을 수행한다. 화면에는 Sphero Ball의 색과 현재 움직임을 표시해주고 충돌을 감지하면 시작 지점으로 돌아온다.

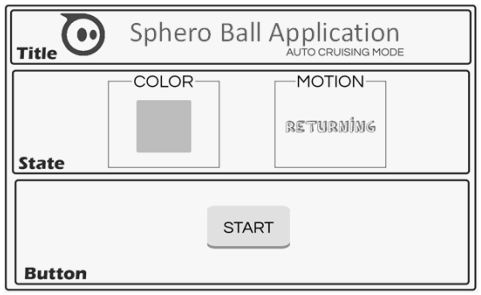


Fig. 7. Auto Mode of Sphero Ball Application

지상용 IoT 애플리케이션으로 제어할 디바이스로 Sphero Ball을 선정한 이유는 공 모양의 형태이므로 바퀴가 없는 디바이스이기 때문에 표면 혹은 환경에 제약이 커 신뢰성 있는 애플리케이션 개발에 제한이 있다. 또한, 제어 시 다른 지상용 디바이스보다 센서가 부족하고 추가적인 탑재가 제한적이므로 이슈들을 더 포함할 수 있도록 선정하였다.

5.2 신뢰도 향상 기법이 적용된 사례

본 절에서는 지상용 IoT 애플리케이션 개발 시 실제로 적용된 사례들을 나열한다. 개발 시 가장 신뢰성을 저해한 상황은 다음과 같은 세 가지였고, 이를 해결하기 위해 적용된 향상 기법들을 설명한다.

1) 오류 관리성 관련 사례

▪ 충돌 감지 오판단 이슈 : Sphero Ball이 이동할 때, 속도가 빠른 상태에서 급격한 회전을 하였더니 충돌로 감지하였고 리턴 기능이 수행되는 현상을 발견하였다. 또한, 속도가 느린 상태에서 벽에 약하게 충돌하면 감지하지 못하는 현상을 발견하였다. Sphero Ball은 공 형태로 되어있기 때문에 충돌 시 디바이스 내부 축의 변화량을 활용하는데 디바이스가 이동할 때 이런 특징을 고려하지 않기 때문에 충돌로 인한 오류를 제대로 처리하지 못하였다.

▪ 적용된 충돌 감지 오류 관리성 향상 기법 : Sphero Ball의 충돌 감지 기능 구현 시 발생한 이슈를 해결하기 위해 4.1절의 충돌 관련 오류 관리성 향상 기법을 적용하였다. 충돌로 오판단하는 상황을 급격한 방향 전환과 속도 변화 두 가지로 분류했다. 그리고 실제 충돌을 감지하기 위한 내부 충격량을 충돌하는 각도 30도, 60도, 90도로 분류하여 추출하였다.

그다음, 급격한 방향 전환 시 내부 충격량값을 추출하고 속도 변화 시 값을 추출하였다. 이 값들을 이용하여 충돌 감지가 발생했을 때, 각 각도에 대한 내부 충격량값과 비교를 하도록 구현하였다. 그리고 패턴 매칭이 되면 충돌로 감지 후 시작 지점으로 돌아가도록 처리하였고, 패턴 매칭이 되지 않으면 해당 이벤트를 무시하도록 구현했다.

제시한 알고리즘 설계를 활용하여 Sphero Ball 이동 시 장애물과의 충돌을 감지하는 기능을 수행한 결과, 실제 충돌로 감지하는 확률이 더 높아졌다.

2) 성숙도 관련 사례

▪ 표면 환경으로 인한 오차 이슈 : Sphero Ball 이동 시 정확한 위치를 알기 위해서 좌표 정보를 활용하는데, 이때 좌표 정보에 오류가 발생하는 현상을 관찰하였다. 이런 현상은 표면의 재질에 따라서 발생하거나 급격한 속도 혹은 방향 전환 시 디바이스가 제자리에서 헛돌게 되어서 발생했다. 이로 인해, 디바이스의 위치는 그대로지만 좌표 정보가 바뀌게 되어 정확한 지점으로 찾아가는 기능에 정확도 오차가 발생하였다.

▪ 적용된 오차 향상 기법 : Sphero Ball의 공회전으로 인한 오차를 보정하기 위해 3.2절의 표면 환경의 성숙도 향상 기법을 적용하였다. 먼저, Sphero Ball 애플리케이션이 실행되면 주변의 지형을 자동으로 calibrate를 수행하도록 하였다. Sphero Ball은 속도별 calibrate를 수행하여 기준값을 정하였다. 다음 Table 1은 초기 좌표 (0, 0)을 기준으로 calibrate를 30cm/s 속도로 6초 동안 수행하여 얻은 좌표값으로 이 속도의 좌표값을 기준으로 마찰력을 1이라 가정하였다.

Table 1. Experimental Calibration Values

	X	Y
1	50.5	52.3
2	49.7	50.1
3	53.1	52
4	50.3	49.7
5	50.1	50.4
6	49.8	51.7
7	48.8	50
8	50.5	56.2
Criteria	50.35	51.59

위의 값을 기준으로 애플리케이션 실행 시 Sphero Ball은 30cm/s 속도로 6초 동안 움직일 때마다, 한 번씩 좌표값이 올바른지 체크해주도록 구현하였다. 체크 시, Fig. 4와 같이 같은 시간에 같은 거리를 갔을 때, 기준 좌표값과 차이가 큰 경우 공회전이 발생하였다고 생각하고 측정된 좌표값에 오류가 있다고 판단하여 보정해주었다. 또한, 같은 시간 내에 같은 거리를 가지 못하면 마찰력이 크다고 판단하고, 좌표값은 변경시키지 않지만 속도의 세기를 늘려 비슷한 결과를 수행하도록 구현하였다.

표면에 경사가 있는 경우 경사가 높으면 명령된 방향과 역으로 변경될 경우이므로, 이때 빗면에 가해지는 힘보다 더 큰 속도를 올리도록 구현하였고 어느 정도 한계는 있었지만 완만한 경사에서는 해결할 수 있었다.

▪ **이동 시 정확도 오류 이슈 :** Sphero Ball 충돌 시 처음 위치로 돌아오는 기능을 구현하였는데, 구현 테스트 시 처음 위치로 정확히 돌아오지 않는 문제가 발생하였다. 다음 Fig. 8은 실제 테스트 시 Sphero Ball의 도착 지점과 처음 위치를 비교한 그림이다.

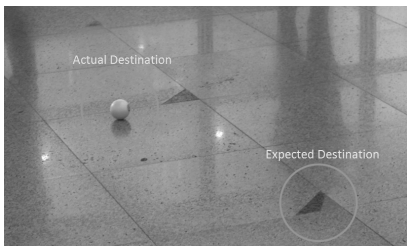


Fig. 8. Situation of Error Distance

위 그림은 실제 Sphero Ball의 처음 위치로 돌아오는 기능 수행 후의 모습이다. 그림과 같이, Sphero Ball이 멈춘 위치와 처음 위치 사이의 오차 거리가 약 1m이었다. 당시 Sphero Ball의 drive(heading, velocity) API를 사용하여 방향과 속도를 입력하고 돌아오도록 하였다. 그러나 제동 시 발생하는 관성에 의한 오차와 방향 전환 시 발생하는 오차 등으로 인해 최종 도착 지점과 처음 위치와 오차 범위가 커지게 되어 애플리케이션의 신뢰성을 저해하였다.

▪ **적용된 정확도 오류 향상 기법 :** 특정 지점 이동 시 정확도에 대한 이슈는 3.1절의 알고리즘을 적용하여 해결하였다. Sphero Ball은 자체적으로 현재의 위치 정보를 제공하기 때문에 해당 정보를 알고리즘에 사용하였다. 이에 따라 목표 지점으로 돌아가는 기능 수행 시, 목표 지점과의 거리를 지속적으로 계산하고, 방향 전환 시 발생하는 관성에 의해 방향이 틀어지면, 알고리즘을 통해 방향을 다시 계산함으로써 정확도를 높일 수 있었다. 다음은 실제 구현한 이동 정확도를 높이는 알고리즘 구현 코드이다.

```
1  public void returnBall()
2  {
3   double criterionDistance = Math.sqrt
      (Math.pow(lastX, 2) + Math.pow(lastY, 2));
4     double criterion = criterionDistance * 0.1
5     criterionDistance *= 0.4
6     double remainingDistance = 0,
        preRemainingDistance = 0;
7     double speed = INIT_SPEED;
8     statusManager.setSpeed(speed)
9  while(remainingDistance > STOPPING_SCOPE) {
10       preRemainingDistance = remainingDistance;
11       remainingDistance = Math.sqrt
       (Math.pow(lastX, 2) + Math.pow(lastY, 2));
12       if(remainingDistance < criterionDistance) {
13          criterionDistance -= criterion;
14          speed *= 0.2f;
15          statusManager.setSpeed(speed);
16       }
17  if(remainingDistance > preRemainingDistance) {
18          angle 1= calculateAngle();
19          statusManager.setHeading(angle);
20       }
21    }
22  }
```

위와 같은 알고리즘을 적용하여 Sphero Ball의 최종 도착 지점의 오차율을 큰 폭으로 줄일 수 있었다. 다음 그림은 실제 알고리즘 적용 후의 도착 결과 그림이다. Fig. 9와 같이 알고리즘 적용 후에 오차가 큰 폭으로 줄어든 것을 볼 수 있었다.

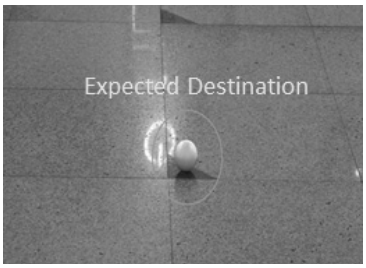


Fig. 9. Reduced Error Distance after Applying the Algorithm

6. 결 론

지상용 IoT 디바이스는 각 디바이스의 특성과 여러 환경 요소에 따라 제어 시 민감하게 영향을 받는다. 특히 지상용 디바이스이기 때문에 발생하는 공회전 문제나, 이동 오차 문제 등으로 인해 안정적이고 신뢰도 있는 애플리케이션 개발이 어렵다. 그러나 기존 연구의 경우, 다양한 지상용 IoT 디바이스에 대한 특성과 구동 환경 때문에 애플리케이션의 신뢰성을 향상시킬 수 있는 솔루션이 부족하다.

본 논문에서는 지상용 IoT 애플리케이션 개발 시 발생할 수 있는 디바이스 이동, 표면 환경, 충돌, 네트워크와 관련된 신뢰성을 저해시킬 수 있는 이슈를 제시하고 해당 이슈들에 대한 향상 기법들을 제시하여 개발 도중 해당 이슈들을 직면하는 경우 제시한 솔루션을 기반으로 문제 해결에 도움을 준다. 마지막으로, 실제 지상용 IoT 디바이스 중 하나인 Sphero Ball을 이용하여, 애플리케이션 개발에서 발생한 신뢰성 관련 이슈를 제시된 향상 기법으로 해결하고, 이를 사례연구를 통해 연구의 실효성을 검증한다.

제시된 신뢰도 향상 기법을 숙지 및 적용하면, 지상용 IoT 애플리케이션 개발 시 발생할 수 있는 다양한 이슈와 그에 따라 발생하는 품질 저하 등의 문제를 예방할 수 있고,

직면한 이슈를 효과적으로 해결할 수 있다. 즉, 효율적인 애플리케이션 개발이 가능하고, 이슈 발생 예방 및 발생 시 효과적으로 해결하여 전체 품질 향상을 기대할 수 있다.

References

[1] Jayavardhana, G., Rajkumar, B., Slaven, M., and Marimuthu, P., "Internet of Things(IoT): A vision, architectural elements, and future directions," *Journal of Future Generation Computer Systems*, Vol. 29, Issue 7, pp.1645-1660, Sept., 2013.

[2] ISO, ISO IS 9126-1: "Software Engineering—Product Quality —Part 1—Quality Model," International Organization for Standardization Geneva, Switzerland, 2001.

[3] Sphero [Internet], http://www.gosphero.com/sphero-2-0/

[4] S. J. Kim, D. H. Shin, J. J June, J. M. Seong, C. W. Park, J. Y. Lee and S. D. Kim, "Technical Challenges in Developing Sphero Ball IoT Applications," *In Proceedings of The 2014 Fall Conference of KIPS*, Nov., 2014.

[5] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of Things: Vision, Applications, and Research Challenges," *Ad Hoc Networks*, Vol. 10, No. 7, pp.1497-1516, Sep., 2012.

[6] H. J. La, S. D. Kim, "Unconventional Issues and Solutions in Developing IoT Applications," *KIPS Tr. Comp. and Comm. Sys.*, Vol.3, No.10, pp.337-350, Mar., 2014.

[7] J. P. Marsman, "Sharing Control with a Robotic Ball," *Radboud University Bachelor Thesis,* pp.1-53, Aug., 2013.

[8] Alsindi, N., Pahlavan, K., and Alavi, B., "An error propagation aware algorithm for precise cooperative indoor localization," *In Proceedings of Military Communications Conference (MILCOM 2006)*, pp.23-25, Oct., 2006.

[9] Joon Young J., Dong Oh K., and Chang Seok B., "Peer to Peer Signal Characteristic Between IoT Devices for Distance Estimation," *In Proceedings of Internet of Things(WF-IoT)*, pp.208-211, Mar.,2014.

[10] Y. K. Chen, "Challenges and Opportunities of Internet of Things," *In Proceedings of the 17th Asia and South Pacitic Design Automation Conference(ASP-DAC 2012)*, pp.383-388, Jan., 2012.

[11] M. H. Kim, M. H. Lee, D. W. Lee, and H. M. Whang, "Study of Acceleration Control on effectiveness for Curve Driving of Torque Driving Apertures," *In Proceedings of 2011 Information and Control Symposiu*, pp.69-73, Apr., 2011.

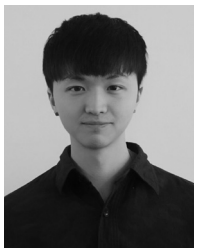
[12] Lego Mindstorm [Internet], http://mindstorms.lego.com/

[13] TurtleBot [Internet], http://whttp://www.turtlebot.com/

[14] Jumping Sumo [Internet], http://www.parrot.com/products/jumping-sumo/

[15] L. Westfall, "The Certified Software Quality Engineer Handbook," Quality Press, Sep., 2009.

[16] Yuxi Liu, Guohui Zhou, "Key Technologies and Applications of Internet of Things," *In Proceedings of 2012 the 5th International Conference on Intelligent Computation Technology and Automation(ICICTA)*, pp.197-200, Jan., 2012.



신 동 하

e-mail : shindh159@gmail.com

현 재 숭실대학교 컴퓨터학부 학사과정

관심분야 : 사물 인터넷 컴퓨팅(Internet of Things Computing), 컨텍스트 인지 컴퓨팅(Context-Aware Computing)

한 승 호

e-mail : gks8030@gmail.com

현 재 숭실대학교 컴퓨터학부 학사과정

관심분야 : 사물 인터넷 컴퓨팅(Internet of Things Computing), 컨텍스트 인지 컴퓨팅(Context-Aware Computing)

김 수 동

e-mail : sdkim777@gmail.com

1984년 Northeast Missouri State University 전산학(학사)

1988년~1991년 The University of Iowa 전산학(석사/박사)

1991년~1993년 한국통신 연구개발단 선임연구원

1994년~1995년 현대전자 소프트웨어연구소 책임연구원

현 재 숭실대학교 컴퓨터학부 교수

관심분야 : 객체지향 모델링(Object-Oriented Modeling), 소프트웨어 아키텍처(Software Architecture), 컨텍스트 인지 서비스(Context-Aware Service), 사물 인터넷 컴퓨팅(Internet of Things Computing)

허 진 선

e-mail : annie8js@gmail.com

2001년 숭실대학교 컴퓨터학부(공학사)

2003년 숭실대학교 컴퓨터학과 (공학석사)

2009년 숭실대학교 컴퓨터학과 (공학박사)

현 재 (주)스마티랩 연구원

관심분야 : 품질 관리(Quality Management), 사물 인터넷 컴퓨팅(Internet of Things Computing), 재사용 공학(Reuse Engineering)