이동로봇의 Herding 문제 적용

Application of Herding Problem to a Mobile Robot

강 민 구, 이 진 수*
(Min Koo Kang and Jin Soo Lee)

Abstract : This paper considers the application of mobile robot to the herding problem. The herding problem involves a 'pursuer' trying to herd a moving 'evader' to a predefined location. In this paper, two mobile robots act as pursuer and evader in the fenced area, where the pursuer robot uses a fuzzy cooperative decision strategy (FCDS) in the herding algorithm. To herd evader robot to a predefined position, the pursuer robot calculates strategic herding point and then navigates to that point using FCDS. FCDS consists of a two-level hierarchy: low level motion descriptors and a high level coordinator. In order to optimize the FCDS, we use the multi-thread evolutionary programming algorithm. The proposed algorithm is implemented in the real mobile robot system and its performance is demonstrated using experimental results.

Keywords : herding, fuzzy system, mobile robot, evolutionary programming

I. 서론

Herding 이란, '추적자(pursuer)'가 '회피자(evader)'를 정해진 곳으로 안전하게 몰아가는 것을 말하며, 양치기 개가 양들을 몰아가는 경우를 예로 들 수 있다. 이때 추적자는 임의의 방향으로 움직이려 하는 회피자를 몰아서 안전한 곳으로 몰아야 한다. 일반적으로 대상을 몰아간다는 것은 회피자와 추적자의 위치, 그리고 몰아가야 할 위치 등의 공간상의 분포에 관한 문제로 볼 수 있다. 회피자를 원하는 방향으로 몰아가기 위한 추적자의 위치를 계산해낼 수 있다면 추적자가 그 위치로 이동함으로써 herding 문제를 해결할 수 있다. 실용적인 관점에서는, 이동 로봇과 herding 문제 자체는 큰 연관성 있는 중요도를 가지지 않을 수 있지만, herding 문제를 로봇에 적용하여 푸는 과정에서 필요한 기술들은 다른 이동로봇의 응용에도 쉽게 적용이 가능한 이동로봇의 기본 기술들이라고 볼 수 있다. 따라서, 현재의 이동로봇에 필요한 기본 기술들을 응용하여 herding 문제를 푸는 과정들은 이동로봇의 다른 응용에 바로 적용이 가능한 기술들을 보여준다고 할 수 있다.

Herding 문제는 일종의 추적/회피 문제라 할 수 있는데, 일반적인 추적/회피 문제는 결과의 측면에서 보면 추적자와 회피자의 최종 위치가 같다는 것이다[1][2]. 즉, 추적자가 회피자를 완전히 따라잡는 것이 최종 목표이다. 한편, herding 문제의 최종 목표는 회피자만을 원하는 상태로 만드는 것인데, 그 과정에서 추적자는 회피자와의 직접적인 접촉이 발생하지 않게 해야 한다. Kachroo *et al.* [3]은 herding 문제를 푸는데 있어서 회피자를 수동적으로 움직이는 대상으로 보고 dynamic programming 기법을 이용했다. 그 후 그들은 [4]에서 [3]의 연구를 확장시켜 회피자를 임의의 확률적인 행동을 하는 대상으로 보았다. 한편, Akbarzadeh *et al.* [5]는 전문가 시스템에 기초한 협동 학습을 이용하는 퍼지 적응 알고리즘을 제안하고 그것을 herding 문제에 적용하였다.

이 논문에서는, herding 문제에 이동로봇을 이용하였다. 이동로봇을 위한 herding 알고리즘을 제안하고 그것의 성능을 실험적으로 보이고자 한다. 정해진 작업 공간 안에서 하나의 이동로봇은 추적로봇이 되고 다른 하나는 회피로봇이 되어, 추적로봇이 회피로봇을 정해진 곳으로 몰아가는 역할을 한다. 회피로봇은 장애물을 피하면서 작업공간의 다른 지점으로 가려는 동작을 하는데, 이것은 '회피자'의 일반적인 특성이며, 회피로봇 입장에서는 추적로봇은 단순한 장애물로 인식될 뿐이다. 추적로봇은 움직이는 회피로봇을 충돌 없이 원하는 지점으로 몰아가야 한다. 이렇게 몰아가는 작업은 상대적이라 할 수 있고, 회피로봇이 항상 똑같이 추적로봇의 움직임에 반응하리라는 보장도 없으며 따라서 회피로봇을 쫓아서 몰아가다가도 그것이 추적로봇 쪽으로 무리하게 다가오면 그것을 피할 수도 있어야 한다. 즉, 회피로봇은 [4]에서 말하는 확률적인 임의의 동작을 한다고 볼 수 있으며 그것을 예측하기란 매우 힘들다. 이 논문에서 제안하는 추적로봇을 위한 herding 알고리즘은, 회피로봇의 위치와 몰아가야 할 위치를 이용해서 추적로봇이 위치해야 하는 장소(Strategic Herding Point : SHP)를 계산해 내고, 장애물과의 충돌 없이 그곳으로 신속하게 주행하는 방식으로 이루어져 있다. 따라서 알고리즘은 두 개의 부분으로 나눠볼 수 있는데, 하나는 herding을 하기 위해서 추적로봇이 가야 할 위치인 SHP를 계산하는 부분이고, 다른 하나는 SHP로 주행하는 알고리즘이다. Herding 문제에서는 장애물을 피하면서 최단 시간에 도착하는 동작이 필요할 뿐만 아니라, 그 주행이 궁극적으로 회피로봇을 원하는 지점으로 몰아가는 역할을 해야 한다. 이런 주행 알고리즘으로는 퍼지를 바탕으로 한 협력결정 전략(Fuzzy Cooperative Decision Strategy: FCDS)을 이용하였다. FCDS는 구조상 디버깅이 쉽게 가능하며, 퍼지 계수들을 진화연산으로 최적 값을 찾기 때문에 원하는 기능을 수행하는 최적의 퍼지 시스템을 구성할 수 있다. 만약 일반적인 퍼지 시스템을 사용한다면 각 퍼지 계수를 찾는 일은 많은 시간이 걸리는 일이 될 것이다. FCDS 는 두 단계로 구성된 퍼지 논

* 책임저자(Corresponding Author)
논문접수 : 2004. 2. 12., 채택확정 : 2004. 10. 18.
강민구, 이진수 : 포항공과대학교 전자전기공학과
(mkkang@postech.ac.kr / jsoo@postech.ac.kr)

리 시스템이다. 하위의 퍼지 추론 모듈(Fuzzy Inference Module: FIM)들은 퍼지 규칙들로 기술된 시스템의 기본적인 행동 모듈들이고, 상위의 퍼지 코디네이터(Fuzzy Coordinator: FC)는 그 행동들을 조율하여 일반적인 동작을 수행할 수 있게 한다. FC는 Takagi-Sugeno 퍼지 추론 기술 [6]을 이용하는데, 그것의 출력은 FIM 출력들의 선형 조합이다. FC에 사용된 규칙들은 예상하기 힘든 환경에서의 주행과 herding 동작을 표현하는 것들이기 때문에 정하기가 쉽지 않다. FC 규칙들의 최적 값을 정하기 위하여 다중 스레드 진화연산 기법(Multi-thread Evolutionary Programming: MEP) [7]을 이용하였는데, MEP는 상황에 따라서 변이연산자들을 선택하여 이용함으로써 빠르게 최적 값을 찾을 수 있는 기법이다.

이 논문의 구성은 다음과 같다. 먼저, 이동로봇을 이용해서 herding을 하기 위한 시스템 환경을 정의하고 herding 알고리즘에 대해 설명한다. 아울러 그 알고리즘에 사용된 FCDS 에 대해서도 다룬다. 마지막에는, 제안된 알고리즘을 실제 로봇에 적용하고 그 결과를 통해서 성능을 확인하고자 한다.

II. Herding 문제와 시스템 환경 정의

Herding은 양치기 개가 양들을 몰아가는 것처럼, 대상과의 직접적인 마찰 없이 대상을 원하는 곳으로 이동시키는 것을 일컷는다. 여기서는 이동로봇에 herding 알고리즘을 구현하는데, 그 대상 역시 다른 이동로봇으로 선택하였다. 즉, 추적자도 이동로봇이고 회피자 역시 이동로봇인데, 두 로봇 모두 unicycle 타입이다.

그림 1은 작업 환경을 보여주는데, 가로 세로 3m 의 울타리 안에 추적로봇과 회피로봇이 각각 하나씩 있다. 추적로봇은 회피로봇을 물리적인 접촉 없이 왼쪽 아래 구석으로 몰아가야 한다. 여기서 회피로봇은 단순하게 장애물을 피해서 작업공간의 정 중앙(그림1에서 + 표시 부분)으로 가려는 동작만을 한다. 즉, 회피로봇은 장착되어 있는 초음파센서와 적외선센서를 이용하여 벽이나 추적로봇 등의 장애물을 감지하여 부딪힘을 피하고 정 중앙으로 움직이려고 한다. 이것은 VFH 기법 [8]을 수정하여 구현하였고, 회피로봇에 대한 자세한 내용은 이 논문에서는 논외로 한다.

추적로봇은, 정 중앙으로 가려는 회피로봇을 몰아서 왼쪽 아래 구석으로 몰아가야 한다. 만약 추적로봇이 다가가면 회피로봇은 그것을 단순히 장애물로만 인식하고 피하려는 동작을 한다. 이것을 이용해서 추적로봇은 회피로봇에 접근하여 구석으로 몰리게끔 하는 것이다. 회피로봇이 정 중앙으로 가게끔 해놓은 설정은 추적로봇이 해야 하는 일인 구석으로 몰아가는 것과 반대로 움직이게 해놓은 가정일 뿐이다. 회피로봇이 어떤 일을 하든지 추적로봇을 피하여 자신의 안전한 길을 택하는 행동만 한다면 추적로봇은 회피로봇을 구석으로 몰아갈 수 있다. 이때 추적로봇은 회피로봇과 직접적인 충돌이 있으면 안 되는데, 이를 위해 추적로봇은 작업공간 내에서의 자신의 위치와 회피로봇의 위치를 정확히 파악할 수 있어야 한다. 추적로봇 자신의 기본적인 위치파악은 dead reckoning [9]으로 구현이 가능하지만 dead reckoning 은 오차가 쉽게 생길 수 있고 그것이 무한대로 커질 수 있는 단점이 있다. 추적로봇은 dead reckoning으로 자신의 위치를 계산하고, 2-D 레이저 센서를 이용해서 벽을 검출하여 자신의 위치를 보정한다. 뿐만 아니라, 그 센서를 이용해서 회피로봇의 위치도 파악하는데, 이것은 모두 화상 처리 기법의 하나인Hough 변환 알고리즘을 이용한 것이다[10]. Hough 알고리즘은 투표 방식으로 원하는 값을 찾는다. 추적로봇의 위치를 보정하기 위해서, Hough 알고리즘으로 벽 정보를 얻어낸다. 벽은 직선으로 표현되고, 레이저 센서에 의해 검출되는 벽은 최소한 2개이므로 벽의 위치를 알 수 있고, 역으로 추적로봇의 위치를 보정해줄 수 있다. 따라서, Hough 알고리즘에서는 직선의 식에 해당하는 매개변수를 얻어낸다. 회피로봇의 위치를 알아내기 위해서는 회피로봇에 의해 나타나는 반원 모양의 레이저 검출 데이터를 이용한다. 이번에는 직선이 아닌 반원의 형태이고, 그 크기는 회피로봇의 크기므로 원의 중심에 대해 투표하는 방식으로 Hough 알고리즘을 실행한다. Hough 알고리즘은 임의로 정해놓은 분해도로 원하는 값을 정하는 것이고, 그 분해도가 작을수록 찾는 시간이 오래 걸린다. 여기서는 분해도를 충분히 작게 하면서 찾는 시간을 줄이기 위해서 그 해가 있을 만한 영역만을 검색하는 방식을 이용하였다. 즉, 직선에 대한 식은 그 찾는 매개변수가 직선이 기울어진 각도와 직선이 원점과 떨어진 거리인데, 각도의 0도 아니면 90도 이고 거리는 0아니면 3m 이다. 따라서 이 주위만을 검색하면 원하는 해를 찾을 수 있다. 회피로봇의 경우는 이전의 위치에서 많이 떨어진 위치에서는 회피로봇이 있을 확률이 거의 없기 때문에 이전 위치 주위만 검색하면 그 해를 빠르게 찾을 수 있다. 자세한 알고리즘은 [12]를 참조하기 바란다.

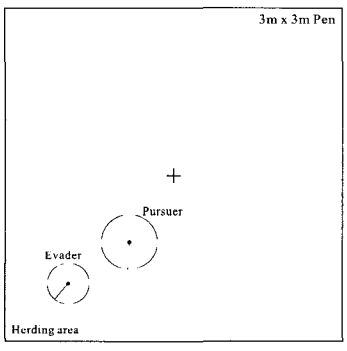


그림 1. 가로 세로 3m의 울타리로 된 작업 환경.
Fig. 1. The herding environment, 3m by 3m pen.

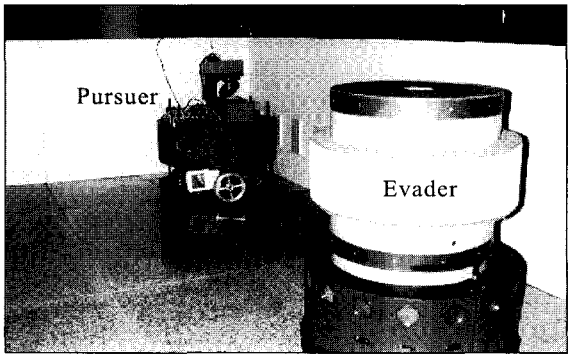


그림 2. 울타리 내의 추적로봇과 회피로봇.
Fig. 2. Robots 'pursuer' and 'evader' in a pen.

그림 2는 작업 환경 내에 있는 두 이동로봇을 보여주고 있다. 추적로봇은 2-D 레이저 센서를 사용하고, 회피로봇은 초음파센서와 적외선 센서를 사용한다. 회피로봇 위에 놓여진 원통은 추적로봇의 2-D 레이저 센서로 회피로봇의 위치를 탐지하기 위한 것이다.

III. 퍼지 협력결정 전략 (FCDS)

FCDS는 하위에 정의된 기본적인 행동들을 상위에서 조율하여, 원하는 고차원적인 행동을 하게 하는 두 단계 형태의 퍼지 시스템이다.

그림 3의 구조를 가지는 FCDS는 다음과 같이 세 단계로 설계된다.

- 1단계 : 기본 행동들을 정의하고 각 행동 모듈을 퍼지 규칙으로 설계
- 2단계 : 기본 행동 모듈을 퍼지 코디네이터로 조율
- 3단계 : 퍼지 코디네이터 매개변수들의 최적화

1단계에서는 로봇의 기본 행동들을 정의한다. 주어진 입력에 대해서 나올 수 있는 모든 퍼지 규칙들을 각 기본 행동들에 따라 분류하여 FIM들을 만든다. 따라서 각 FIM은 정의된 기본 행동에 적합한 퍼지 규칙들의 집합이라고 볼 수 있다. 일반적으로 FIM 의 l번째 퍼지 *IF-THEN* 규칙은 다음과 같이 기술된다:

$$R^l : IF\ d_1 \text{ is } F_1^l,\ d_2 \text{ is } F_2^l, \cdots, \text{and } d_n \text{ is } F_n^l,\ THEN\ V_o^l \text{ is } G_R^l \quad (1)$$

여기서 $d_1, d_2, \cdots, d_n$ 은 입력 변수들이고, V_o^l 은 출력 변수다. 그리고 $F_1^l, F_2^l, \cdots, F_n^l$ 은 입력 변수들에 대한 퍼지 집합들이고, G_R^l 은 출력변수에 대한 퍼지 집합이다. 이와 같은 규칙들에 의해 각 FIM들은 입력변수에 따라 고유의 출력 값을 만들어 낸다. 만약 어떤 FIM에 M개의 규칙이 있고, singleton 퍼지화기(fuzzifier)와 곱셈 추론, 그리고 평균중심법 비퍼지화기(defuzzifier)를 사용한다면 그 출력은 다음과 같다[11].

$$V_o = \sum_{l=1}^{M} \Theta^l \Gamma^l = \sum_{l=1}^{M} \Theta^l \left(\frac{\prod_{i=1}^{n} \mu_{F_i^l}(d_i)}{\sum_{l=1}^{M} \prod_{i=1}^{n} \mu_{F_i^l}(d_i)} \right) \quad (2)$$

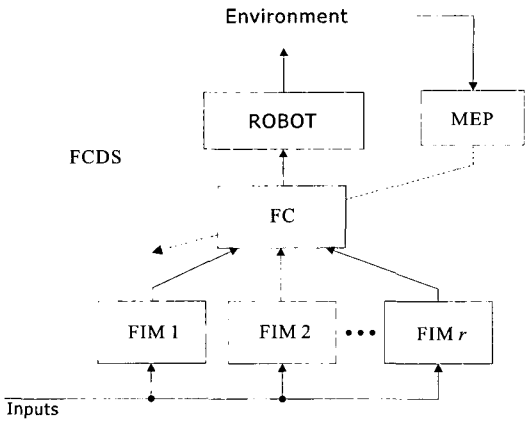


그림 3. 퍼지 협력결정 전략의 구조.
Fig. 3. The architecture of a FCDS.

여기서 $\mu_{F_i^l}(d_i)$ 는 입력 d_i 에 대한 퍼지 집합 F_i^l 의 소속함수 (membership function) 값이다. Γ^l 은 l번째 규칙의 퍼지 기저 함수(Fuzzy Basis Function: FBF)이고 Θ^l 은 소속함수 $\mu_{G_R^l}$ 을 최대로 만드는 값이다.

2단계에서는 FC가 각 FIM 출력들을 조율하는데, FC는 그 FIM 출력들의 가중치 합으로 결과를 만들어 낸다. FIM 들은 단순한 기본 행동들을 기술한 모듈들인데 비해, FC는 상위에서 그것들을 조율하는 역할을 해서 결과적으로 복잡한 일을 수행할 수 있게 한다. FC는 Takagi-Sugeno [6] 형식의 퍼지 논리 시스템으로서, 그 퍼지 규칙들은 다음과 같이 기술된다.

$$R^k : IF\ V_{M_1} \text{ is } F_1^k,\ V_{M_2} \text{ is } F_2^k, \cdots, \text{and } V_{M_r} \text{ is } F_r^k,\ THEN\ V_o^k = c_1^k V_{M_1} + c_2^k V_{M_2} + \cdots + c_r^k V_{M_r} \quad (3)$$

여기서 $V_{M_1}, V_{M_2}, \cdots, V_{M_r}$ 은 FIM $1,2,\cdots,r$ 의 출력에 해당하는 입력변수들이고, V_o^k 는 출력변수이다. 그리고 $c_1^k, c_2^k, \cdots, c_r^k$ 는 실수 값의 매개변수들인데, 이 값들을 조정하여 원하는 동작을 수행하게 만들 수 있다. N개의 퍼지 규칙들이 있다면 FC 의 출력 값은 다음과 같이 주어진다.

$$V_o = \sum_{k=1}^{N} V_o^k \Phi^k = \sum_{k=1}^{N} V_o^k \left(\frac{\prod_{p=1}^{r} \mu_{F_p^k}(V_{M_p})}{\sum_{k=1}^{N} \prod_{p=1}^{r} \mu_{F_p^k}(V_{M_p})} \right) \quad (4)$$

여기서 $\mu_{F_p^k}$ 는 퍼지 집합 F_p^k 의 소속함수 값이고, Φ^k 는 k 번째 규칙의 FBF이다.

3단계에서는 FC의 매개변수들 $c_1^k, c_2^k, \cdots, c_r^k, k \in \{1, \cdots, N\}$ 을 MEP [7]로 최적화한다. FIM에서는 기본 행동들에 대한 규칙을 정의했는데, 그것들은 쉽게 분석할 수 있는 규칙들이라서 설계하기가 용이하다. 하지만 FC는 FIM의 출력을 이용하여 고도의 복잡한 일을 수행할 수 있게 하는 규칙들이라 설계가 쉽지 않다. 가령 추적로봇이 회피로봇을 구석으로 몰기 위해서는 회피로봇에 접근해야 하고 이것은 자칫하면 충돌로 이어질 수 있다. 충돌을 회피하며 효과적으로 몰아가는 동작을 하는 퍼지 규칙은 설계가 어렵기 때문에, FC의 매개변수들을 진화연산 기법의 하나인 MEP로 최적화하고자 한다. MEP는 세 가지의 탐색 루틴을 가지고 있는데, 상황에 따라 그것을 적절히 선택해서 사용함으로써 효율적인 최적값 탐색이 가능하다. 뿐만 아니라, 변이 연산자로는 넓은 지역을 탐색할 때 수렴속도가 빠른 Cauchy 연산자와 좁은 지역을 세밀히 탐색할 때 유리한 Gaussian 연산자를 선택적으로 사용하였다. 이와 같이 각 연산자를 탐색 상황에 따라 각각 다르게 선택하여 사용함으로써, 일반적으로 사용하는 Gaussian 연산자만을 사용했을 때보다 빠르게 그 최적 값을 찾을 수 있다는 장점이 있다[7].

IV. FCDS 를 적용한 Herding 알고리즘 구현

여기서는 실제 이동로봇을 이용한 herding 알고리즘 구현과 앞에서 다룬 FCDS를 적용하는 문제에 대해서 다룬다.

Herding은 회피로봇의 위치를 이용해서 가로막을 위치인 SHP를 결정하고 FCDS를 이용해서 그 위치로 주행함으로써 이루어진다. 주행 중에는 회피로봇과의 충돌을 피하는 길로 가야하며 이것이 FCDS의 주 역할이다.

1. 추적로봇의 센서시스템

추적로봇에는 그림 4와 같은 2-D 레이저 센서가 장착되어 있다. 2-D 레이저 센서는 회전 거울을 이용하여 뒤쪽의 90° 을 제외한 나머지 270° 영역 안에 있는 장애물까지의 거리를 알려준다. 한번의 측정에 걸리는 시간은 0.125초이고, 해상도는 약 0.615° 이다. 따라서 그림 4와 같이 매 0.125초마다 440개의 거리 값들, 즉 $l_0, \cdots l_{439}$ 을 제공한다. 추적로봇은 이 거리 값들을 이용해서 회피로봇의 위치도 파악하고 벽을 감지해 자신의 위치를 보정하기도 한다. 회피로봇의 위치는 뒤에서 다룰 SHP를 계산하는 가장 기초적인 값이다. 또한 이 거리 값들은 FCDS의 입력으로 사용된다.

2. SHP의 계산

추적로봇이 위치해야 하는 SHP는 2-D 레이저 센서에 의해 얻어진 회피로봇의 위치 $\mathbf{x}_{evader}$ 을 이용해서 그림5와 같이 구한다.

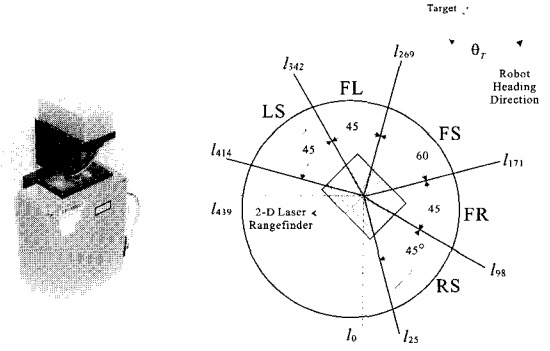


그림 4. 2-D 레이저 센서와 로봇에서의 배치.

Fig. 4. 2-D laser rangefinder and its spatial configuration.

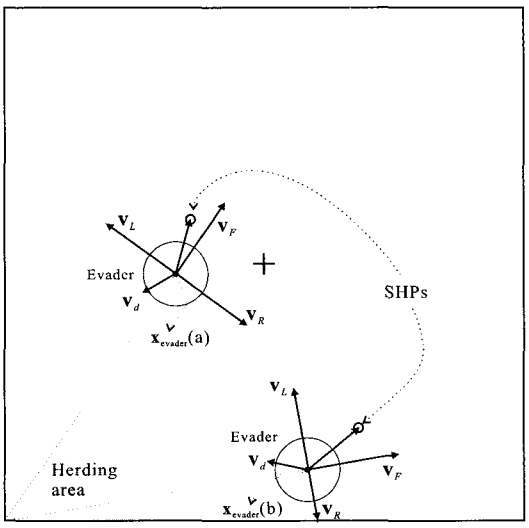


그림 5. SHP의 계산.

Fig. 5. Calculation of SHP.

그림 5는 회피로봇의 두 위치인 $\mathbf{x}_{evader}(a)$ 와 $\mathbf{x}_{evader}(b)$ 에 대한 SHP들을 보여주는데, 그것을 구하는 방법은 다음과 같다. 먼저 몰아가야 할 구석에서 회피로봇으로 연결된 직선, 그리고 그 직선과 수직인 직선이 회피로봇의 중심에 있다고 생각하자. 그림5에서처럼 우선 회피로봇의 중심에서 그 직선들 방향으로 3개의 벡터를 만든다. 벡터 $\mathbf{v}_F$ 는 구석과 정반대 방향으로 만든 벡터이고, $\mathbf{v}_R$ 은 오른쪽으로, $\mathbf{v}_L$ 은 왼쪽으로 만든 벡터이다. 각 벡터의 크기는 다음과 같이 구할 수 있다. 각각의 최대 크기를 K_F, K_L, K_R 이라 하고, 이 때 $K_L = K_R$ 이다. 만약 벡터를 최대 크기로 했을 때 벡터가 울타리를 벗어나게 되면 울타리에 닿는 지점까지만을 그 벡터의 크기로 정한다. 즉,

$$\|\mathbf{v}_{(\cdot)}\| = \begin{cases} d_{(\cdot)}(k) & \text{if vector exists outside the pen,} \\ K_{(\cdot)} & \text{otherwise.} \end{cases} \tag{5}$$

여기서 $d_{(\cdot)}(k)$ 는 회피로봇의 중심과 울타리까지 해당 벡터방향으로의 거리이고, 항상 $K_{(\cdot)}$ 보다는 작게 된다. 그림5에서 회피로봇이 $\mathbf{x}_{evader}(a)$ 에 있는 경우는 각 벡터가 최대값을 가지게 된다. 하지만 $\mathbf{x}_{evader}(b)$ 의 경우는 $\mathbf{v}_R$ 이 최대크기일 때 울타리를 벗어나게 되므로 $\mathbf{v}_R$ 은 울타리까지만을 그 크기로 정한다. 마지막으로 K_d 를 설계 상수라 했을 때, $\mathbf{v}_d$ 는 다음과 같이 계산된다.

$$\mathbf{v}_d = K_d \cdot (\mathbf{x}_{evader}(k) - \mathbf{x}_{evader}(k-1)) \tag{6}$$

$\mathbf{v}_d$ 는 회피로봇의 예상 진로 방향으로의 벡터라고 볼 수 있다. 즉 회피로봇의 이전 위치와 현재 위치를 이용해서 예상 진로와 그 속도를 가늠하여 만든 벡터이다. 이렇게 네 벡터를 잡았을 때, SHP는 (7)과 같이 예상된 회피로봇의 위치와 네 벡터의 합이 된다.

$$\text{SHP} = \mathbf{x}_{evader} + \mathbf{v}_F + \mathbf{v}_L + \mathbf{v}_R + \mathbf{v}_d. \tag{7}$$

$\mathbf{x}_{evader}(a)$ 에서는 $\mathbf{v}_L = -\mathbf{v}_R$ 이므로, $\mathbf{v}_F$ 와 $\mathbf{v}_d$ 에 의해서만 SHP가 결정된다. $\mathbf{x}_{evader}(b)$ 에서는 $\|\mathbf{v}_L\| > \|\mathbf{v}_R\|$ 이고 $\mathbf{v}_d$ 역시 $\mathbf{v}_L$ 방향이므로 $\mathbf{v}_F$ 의 왼쪽에 SHP가 놓이게 된다. 효율적으로 회피로봇을 몰고 가기 위해서는 K_F, K_L, K_R, K_d 를 적절한 값으로 설정해야 한다. 만약 K_F 를 너무 작게 잡으면 회피로봇과 충돌할 위험이 커지게 되고, 작게 잡으면 회피로봇이 쉽게 빠져나갈 수 있어서 몰아가는 효과가 떨어지게 된다. K_L, K_R 은 회피로봇이 울타리에 가깝게 위치했을 때의 SHP를 주도적으로 결정하게 되는데, 이 값이 작으면 SHP가 울타리에 가깝게 붙게 되고 따라서 반대 방향으로 회피로봇

이 빠져나갈 가능성이 커진다. 반대로 K_L, K_R 을 크게 잡으면 울타리에서 멀리 떨어지게 되고 회피로봇은 울타리를 따라서 빠져나갈 수 있다. $\mathbf{v}_d$ 는 회피로봇의 예상 방향을 고려해 준 것인데, (6)의 차등 항은 쉽게 측정 잡음을 탈 수 있다. K_d 를 크게 하면 잡음 성분까지 커질 수 있기 때문에 SHP가 안정적으로 얻어지지 않을 수 있다. 만약 K_d 을 너무 작게 설정하면 예상진로 예측 능력이 떨어져 효율적인 herding을 하기가 힘들어진다.

위와 같이 얻어진 SHP는 추적로봇이 가야 할 목표지점이며 FCDS를 이용해서 주행한다. 주행 시에는, 반드시 회피로봇과의 충돌은 피해야 한다.

3. FCDS의 적용

FCDS 의 입력으로 사용하는 값은 다섯 개의 거리 값과 하나의 각도 값이다. 2-D 레이저 센서가 만들어내는 440개의 거리 값들을 구역별로 모아서 다섯 개의 작은 그룹들을 만든다. 즉, *FS*(Front Side), *FL*(Front Left), *LS*(Left Side), *FR*(Front Right), *RS*(Right Side)로 각각 구분된다(그림 4 참조). 각 그룹에 해당하는 거리 값들을 평균하여 다섯 개의 입력 변수 $d_{FS}, d_{FL}, d_{FR}, d_{LS}, d_{RS}$ 를 만드는데, 이들은 다음과 같이 계산된다.

$$d_{FS} = \frac{\sum_{i=171}^{268} l_i}{98}, \quad d_{FL} = \frac{\sum_{i=269}^{341} l_i}{73}, \quad d_{FR} = \frac{\sum_{i=98}^{170} l_i}{73},$$

$$d_{LS} = \frac{\sum_{i=342}^{414} l_i}{73}, \quad d_{RS} = \frac{\sum_{i=25}^{97} l_i}{73}$$

이 다섯 가지 입력 변수들에 대해 두 가지의 퍼지 집합인 NEAR와 FAR를 할당하고, 그 소속함수는 그림 6과 같이 정하였다.

또 다른 하나의 입력 변수로 목표 각도가 있는데, 이것은 SHP 와 추적로봇 간의 각도 차이이다. 즉 로봇의 진행방향과 로봇에서 바라본 SHP로의 방향과의 차이 θ_T 이다(그림 4 참조). θ_T 에 대해서는 다섯 가지의 퍼지 집합들 *F*(Front), *FL*(Front Left), *FR*(Front Right), *L*(Left), *R*(Right)이 할당되고, 그것들의 소속함수들은 그림 6에 나와 있다. 각 FIM들은 출력 값으로 로봇의 각속도 ω 를 만들어 내는데, 이 출력 변수에 대한 퍼지 집합으로는 *ST*(Straight), *LS*(Left Small), *RS*(Right Small), *LB*(Left Big), *RB*(Right Big)가 할당되었고, 그것들에 대한 소속함수들 역시 그림 6에 표시했다.

$d_{FS}, d_{FL}, d_{FR}, d_{LS}, d_{RS}$ 와 θ_T 가 입력 변수들이고, ω 가 출력 변수인 셈이다. 따라서 FIM의 퍼지 규칙 중 한가지 예는 다음과 같이 기술될 수 있다.

R^l : *IF* d_{FS} is *FAR*, d_{FL} is *FAR*, d_{LS} is *NEAR*, d_{FR} is *NEAR*, d_{RS} is *FAR*, and θ_T is *F*, *THEN* angular velocity ω is *LS*.

2개의 퍼지 집합을 가지는 5개의 입력과 5개의 퍼지 집합을 가진 1개의 입력에 의해 나올 수 있는 모든 퍼지 규칙의 종류는 $2^5 \times 5 = 160$ 개다. 이동로봇의 herding 문제 적용에 있어서 기본 행동 모듈인 FIM은 *MF*(Move Forward), *ML*(Move Left), *MR*(Move Right) 세 가지로 정했다. 그리고 각 규칙들은 이 세 가지의 FIM로 각각 의미에 맞게 분배되었다. 가령 위에 예로 든 퍼지 규칙은 *MF*를 위한 FIM에 속한다고 볼 수 있다.

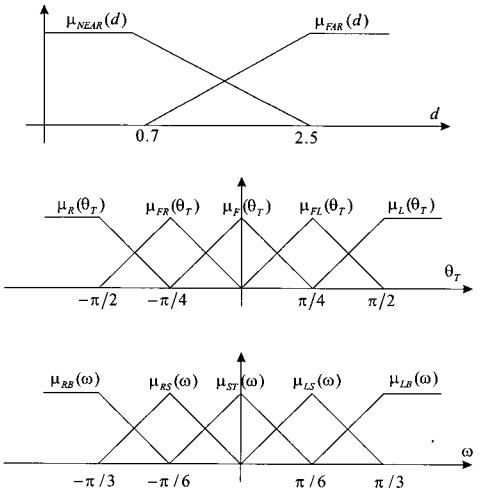


그림 6. FIM 퍼지 집합들의 소속함수들.
Fig. 6. Membership fuctions for FIM fuzzy sets.

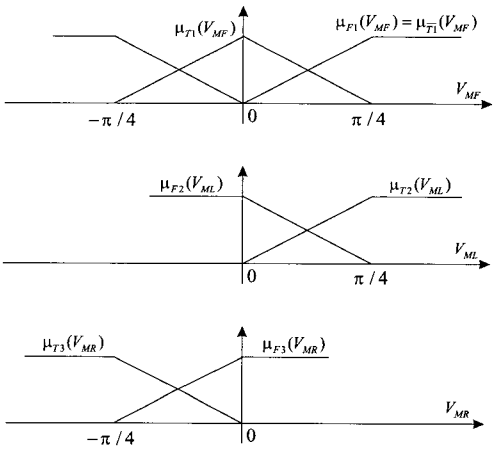


그림 7. FC 퍼지 집합들의 소속함수들.
Fig. 7. Membership functions for FC fuzzy sets.

상위 단계에서는 FC가 하위 단계의 FIM들이 만들어내는 각속도 값들인 V_{MF}, V_{ML}, V_{MR} 을 입력 변수로 받는다. 그 입력 변수들에 대한 퍼지 집합은 *T*(True)와 *F*(False)로 정의하였고, 각 입력 변수들에 대한 소속함수는 그림7과 같다. FC의 퍼지 규칙은 모두 $2^3 = 8$ 개이고, 각 규칙마다 세 개의 매개변수 $c_1^k, c_2^k, c_3^k, k \in \{1, \cdots, 8\}$ 가 있으므로 모두 24개의 매개변수들을 정해줘야 한다. 움직이는 회피로봇과 벽에 부딪히지 않고 SHP로 주행함으로써 herding이 가능하게 하는 24개의 매개변수들을 정하는 문제는 쉽지 않다. 따라서 그 값들을 진화연산 기법의 하나인 MEP로 찾으려 한다.

4. MEP를 이용한 FCDS 최적화

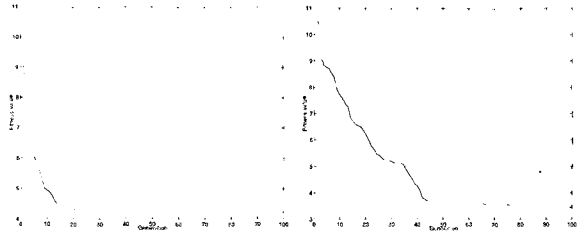


그림 8. 적합도 함수 F의 변화.
Fig. 8. Fitness values for FCDS.

FC를 MEP로 학습시키기 위해 적합도 함수(fitness function) $F = F_d + F_m + F_c$ 을 도입하고, 그것의 요소인 F_d, F_m, F_c 를 다음과 같이 정의했다.

$$F_d = K_d \cdot \frac{1}{T} \cdot \sum_{k=0}^{T} (G(k) + \alpha)^2,$$

$$F_m = \begin{cases} M_{\max} & \text{if collision occurs} \\ K_m \cdot T & \text{otherwise,} \end{cases} \tag{8}$$

$$F_c = \begin{cases} C_{\max} & \text{if } T \geq T_{\max} \\ K_c \cdot \frac{1}{T} \cdot \sum_{k=0}^{T} (\underline{D}(k) + \beta)^{-1} & \text{otherwise.} \end{cases}$$

여기서 T는 회피로봇을 원하는 곳으로 완전히 몰 때까지 걸린 시간이고, $T_{\max}$ 는 주어진 제한 시간이다. $G(k)$ 는 시간 k 에서 추적로봇과 SHP까지의 거리이고, $\underline{D}(k)$ 는 추적로봇과 주위 장애물과의 최단 거리이다. 그리고 K_d, K_m, K_c, $\alpha, \beta, C_{\max}, M_{\max}$ 들은 설계 상수들이다. F_d 는 추적로봇이 얼마나 충실하게 SHP로 잘 주행 했는지를 가리키는 지표이고, F_m 는 회피로봇을 원하는 곳으로 얼마나 빨리 몰아갔는지를 가리키는 지표이다. 마지막으로 F_c 는 추적로봇이 주위의 벽이나 회피로봇에 부딪히지 않고 얼마나 안전하게 주행했는지를 가리키는 지표이다. 지표들의 정의에 따라 F는 추적로봇이 SHP로 빠르게 이동했을 때, 주위 벽이나 회피로봇에 부딪히지 않고 작업을 끝까지 수행했을 때, 그리고 회피로봇을 빠르게 원하는 곳으로 몰았을 때 작은 값을 가지게 된다. FC 매개변수들의 최적 값은 적합도 함수 F를 최소화했을 때 얻어진다. 시뮬레이션을 통해, FC 매개변수들의 최적 값을 찾았고, 그것을 추적로봇에 적용하였다.

그림 8은 두 가지 초기 설정에 대해 시뮬레이션으로 FC의 매개변수들의 최적 값을 찾을 때의 적합도 함수 F의 변화를 보여준다. 이 값은 100개의 개체 중에서 상위 20개의 적합도 함수 F의 평균값을 나타낸 것이다. 그림에서 알 수 있듯이, 약 40세대 내에서 이미 그 최적 값에 거의 도달했음을 알 수 있다.

V. 실험 결과

그림 9은 앞 장의 알고리즘들을 추적로봇에 구현할 때의 전체적인 구조이다.

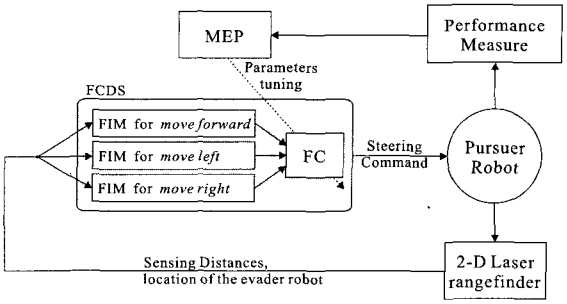
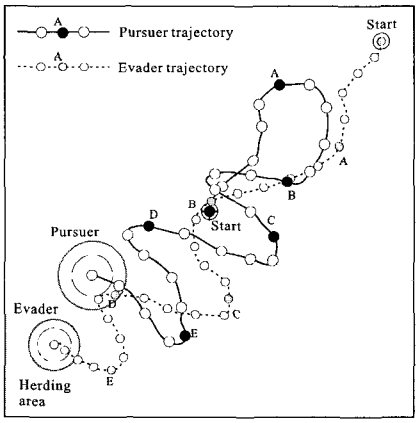
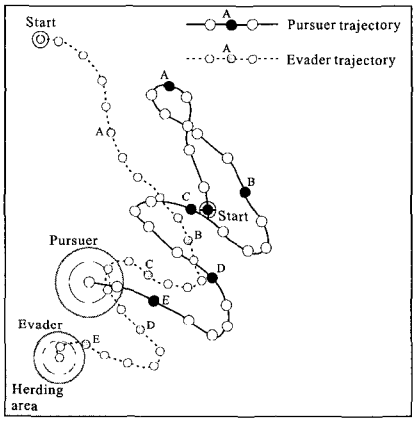


그림 9. 추적로봇의 구조.
Fig. 9. The structure of the pursuer robot.



(a)



(b)

그림 10. 실험 결과.
Fig. 10. Experimental results.

매 0.125초마다 2-D 레이저 센서로부터 거리 값들을 받아 회피로봇의 위치를 파악한다. 그 결과들은 FCDS로 입력되고, FCDS는 로봇의 각속도를 만든다. 이렇게 만들어진 각속도 ω 를 이용하여 선속도 v 는 다음과 같이 정한다.

$$v = V_{\max} \left(1 - \frac{|\omega|}{\omega_{\max}} \right) \tag{9}$$

여기서 $V_{\max}$ 와 $\omega_{\max}$ 는 최대 선속도와 최대 각속도이고, 실험

에서는 각각 $0.4\,m/\sec$ 와 $0.5\pi\ rad/\sec$ 으로 설정했다. 회피로봇의 경우는 추적로봇보다 빠르면 herding이 제대로 되지 않으므로 최대 선속도를 $0.3\,m/\sec$ 로 잡았고, 최대 각속도는 추적로봇의 최대 각속도와 같게 잡았다. SHP에 거의 도착했을 때에는 추적로봇의 선속도를 강제로 줄였다. 이렇게 하지 않으면 추적로봇이 SHP 주위에서 불안정하게 움직일 수 있기 때문이다.

FC매개변수들의 최적 값을 찾기 위해 컴퓨터 시뮬레이션을 실행하였는데, 회피로봇은 몰아가야 할 곳의 반대편 구석에서, 추적로봇은 중앙에서 각각 출발한다. 한번의 시뮬레이션은 추적로봇이 벽이나 회피로봇에 부딪힐 때까지, 또는 성공적으로 회피로봇을 구석으로 몰았을 때까지 진행되었다. 임의로 만든 개체를 수십 세대를 걸쳐 진화시켜서 최적화를 시켰는데, 이렇게 나온 값들 중 적합도가 가장 좋은 경우의 매개변수 값들을 취해서 사용했다.

실제 실험은 그림 2에 보인 이동로봇들로 행해졌다. 회피로봇은 *RWI*사의 상용 이동로봇이고, 추적로봇으로 사용된 것은 자작 로봇이다. 그림 10는 시뮬레이션에 의해 최적화된 FCDS를 실제로 추적로봇에 구현하여 실험한 결과로서, 성공적으로 회피로봇을 구석으로 몰아가는 것을 보여주고 있다.

회피로봇은 항상 작업공간의 중앙으로 가게끔 되어 있으므로 추적로봇이 가로막지 않으면 중앙으로 움직일 것이다. 하지만 추적로봇이 그 길목을 계속 차단하면서 회피로봇을 구석으로 몰아가는 것을 볼 수 있다. 그림10에서 실선은 추적로봇의 경로이고, 점선은 회피로봇의 경로이다. 경로 중간의 원들은 매 1초마다의 위치를 가리키고 매 5초의 위치는 검은 색으로 표시했으므로 두 로봇간의 같은 시간대에서의 위치들을 쉽게 파악할 수 있다. 그림 10(a)는 시뮬레이션과 같은 초기 조건에서 출발한 것이고, 그림 10(b)는 다른 초기 조건에서 출발한 것이다. 하지만 두 가지 모두 성공적으로 회피로봇을 왼쪽 아래 구석으로 몰아가는 것을 보여준다. 특히 두 번째 경우는 시뮬레이션의 초기조건과 다른 초기조건에서도 성공적으로 원하는 herding 작업을 수행할 수 있다는 것을 보여준다.

이 논문에서 다룬 추적로봇과 회피로봇은 한국 지능로봇 경진대회 (http://www.irc.or.kr)에 '양치기 로봇'으로 출품되어 금상을 수상하였다.

VI. 결론

이 논문에서는 이동로봇으로 herding문제를 풀어보았다. 두 개의 이동 로봇이 각각 '추적자'와 '회피자'가 되어 정해진 울타리 안에서 움직이는데, 추적로봇은 퍼지 협력결정 전략(FCDS)을 herding 알고리즘에 이용했다. 회피로봇을 정해진 장소로 몰아가기 위하여 추적로봇이 위치해야 할 전략적인 장소를 구하는데 이 지점을 SHP(Strategic Herding Point)라고 하며 이는 회피로봇의 위치를 이용하여 얻었다. 추적로봇은 SHP로 FCDS를 이용해 주행하는데, FCDS는 두 단계의 계층 구조로 이루어져 있다. 하위 단계에서는 퍼지 행동 모듈(FIM)들이 추적로봇의 기본적인 행동을 기술하는데, move forward, move left, move right라는 세 개의 기본 행동들이 정의되었고, 각각에 대한 FIM들을 구성했다. 상위 단계에서는 퍼지 코디네이터(FC)가 그 기본 행동들을 상황에 맞게 조율하여 고차원적인 일을 수행할 수 있게 하는데, 여기서는 장애물과 부딪히지 않으면서 SHP로 이동함으로써 회피로봇을 구석으로 몰아가게 했다. 또한 그런 herding 동작을 하게끔 하기 위해 다중 스레드 진화연산 기법을 이용하여 FC의 매개변수들을 최적화했다. 제안된 herding알고리즘은 실제 이동로봇에 구현하였고 그 성능을 실험을 통해 보였다.

참고문헌

[1] Basar, Tamer, Olsder, and G. Jan, *Dynamic Noncooperative Game Theory,* 2nd ed. Academic, 1982.

[2] Y. Yavin and M. Pachter, Eds., *Pursuit-Evasion Differential Games,* Pergamon, 1987.

[3] P. Kachroo, S. Shedied, J. Bay, and H. Vanlandingham, "Dynamic programming solution for a class of pursuit evasion problem: The herding problem," *IEEE Trans. Sys., Man, Cybern. C,* vol. 31, pp. 35-41, Feb. 2001.

[4] P. Kachroo, S. Shedied, and H. Vanlandingham, "Pursuit evasion: the herding noncooperative dynamic game – the stochastic model," *IEEE Trans. Sys., Man, Cybern. C,* vol. 32, pp. 37-42, Feb. 2002.

[5] M. Akbarzadeh, H. Rezaei, and M. Naghibi, "A fuzzy adaptive algorithm for expertness based cooperative learning, application to herding problem," *22nd Int. Conf. the North American Fuzzy Information Processing Society,* pp. 317-322, 2003.

[6] M. Sugeno and M. Nishida, "Fuzzy control of model car," *Fuzzy Sets and Systems,* vol. 16, pp. 103-113, 1985.

[7] C. Hong, J. Won, and J. Lee, "Multi-thread evolutionary programming and its application to truch-and-trailer backer-upper control," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Science,* vol. E84-A, pp. 597-603, Feb. 2001.

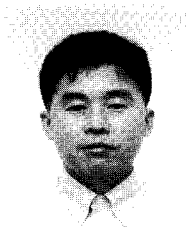
[8] J. Borenstein and Y. Koren, "The vector field histogram fast obstacle avoidance for mobile robot," *IEEE Trans. Robotics and Automation,* vol.7, no.3, pp. 278-288, 1991.

[9] Z. Yu and G. Chirikjian, "Probabilistic models of dead-reckoning error in nonholonomic mobile robots," *IEEE Int. Conf. Robotics and Automation,* vol. 2, pp. 1594-1599, 2003.

[10] D. Pao, H. Li, and R. Jayakumar, "Shapes recognition using the straight line Hough transform: theory and generalization," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 14, pp. 1076-1089, 1992.

[11] L.-X. Wang, *Adaptive fuzzy systems and control,* Prentice hall, 1994.

[12] 강민구, "양치기 로봇(Robot Sheepdog) - Mobile Robot의 응용 예," 반도체네트워크 2002년 6, 7월호, pp. 136-145, 2002.



강 민 구

1991년 서울대학교 제어계측공학과 졸업. 1993년 포항공과대학교 전자전기공학과 석사. 1993년~1999년 LG전자 디지털미디어연구소 선임연구원. 1999년~현재 포항공대 대학원 전자전기공학과 박사과정. 관심분야는 이동로봇제어, 지능제어.

이 진 수

1975년 서울대학교 전자공학과(공학사). 1980년 미국 University of California, Berkeley, Department of Electrical Engineering and Computer Science(M.S.). 1984년 University of California, Los Angeles, Department of System Science(Ph.D.). 1989년~현재 포항공과대학교 전자전기공학과 교수. 관심분야는 적응제어, 지능제어, 로보틱스, 네트웍 제어.