충돌 회피 알고리즘

2021 / 04 / 15

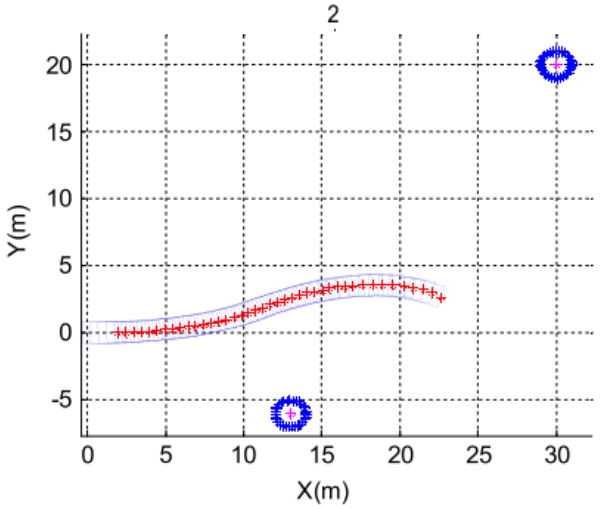
충돌 회피 알고리즘

- 로봇의 운행 경로상에 장애물이 있을 때 회피 기동을 하지 않으면 충돌함

- 장애물과의 충돌은 재산 피해, 인명 피해를 유발함

충돌 회피 알고리즘

- 목적
  - 최종 목적지에 도달
  - 장애물과의 충돌 회피
  - 로봇 운행에 부하가 적은 형태



- 장애물의 종류
  - 정적 장애물
    - ✓ 입간판
    - ✓ 표지판
  - 동적 장애물
    - ✓ 보행자
    - ✓ 차량 등등

충돌 회피 알고리즘

I. 자율주행 충돌회피 알고리즘

- **Control Theory**
  - ✓ **Model based Nonlinear Control**
  - ✓ **Model Prediction Control**

- **Collision Avoidance Planning**
  - ✓ **Grid Map based planning**
  - ✓ **Nearness Diagram navigation**
  - ✓ **Follow the Gap**
  - ✓ **A* (A – Star)**
  - ✓ **Etc. (RRT, D*, …)**

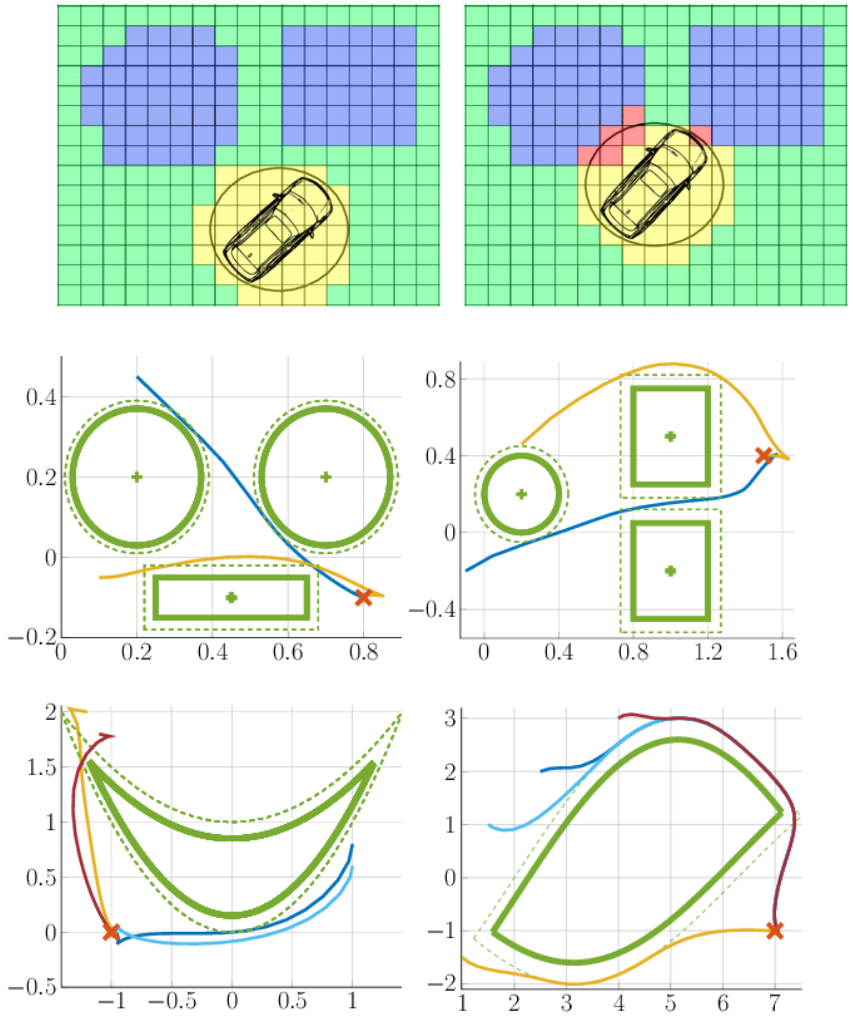
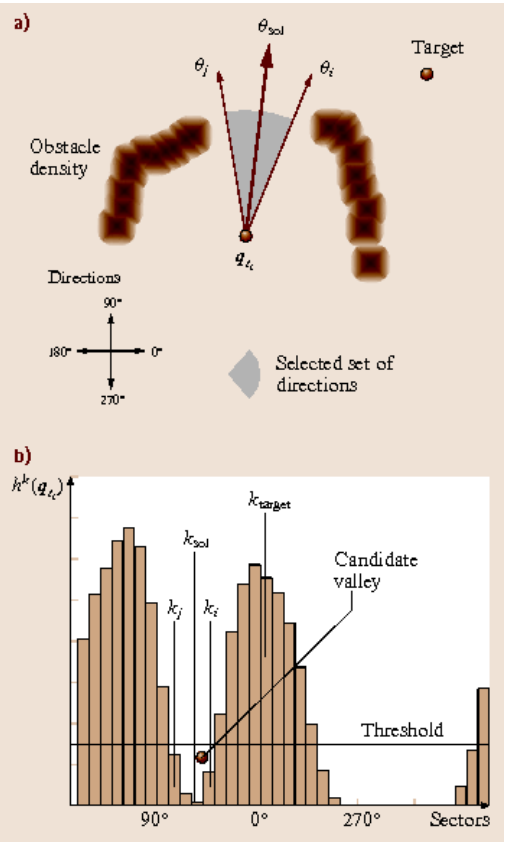


Fig. 3. Four obstacle avoidance scenarios using PANOC. The red x mark



충돌 회피 알고리즘 – 제어기 설계

I. Model based Nonlinear Control

- 장점
  - ✓ 안정도 판별을 통하여 수렴성에 대한 예측이 가능함
  - ✓ 해당 시스템에 대하여 디테일한 설계가 필요하기 때문에 신뢰도가 높음
- 단점
  - ✓ 설계 난이도가 매우 높음
  - ✓ 설계에 따라서 연산량이 많음
  - ✓ 모델 특성에 따라 설계해야 하므로 범용성이 낮음

II. Model Predictive Control

- 장점
  - ✓ 임의의 수학적 Constraint를 고려한 설계가 가능
  - ✓ 설계 난이도가 낮음
  - ✓ Robustness를 고려한 설계가 적용 되어있는 경우 범용성을 충족함
- 단점
  - ✓ 최적화 알고리즘이기에 연산량이 매우 많음
  - ✓ Constraint가 추가 될 수록 실시간성을 보장하기 힘들어 짐
  - ✓ Local Minima 방식의 설계 (Global optimization은 쉽지 않음)

II. Trajectory Tracking and Collision Avoidance

In this section we consider a nonholonomic mobile robot for which we want to design a controller that guarantees asymptotic tracking of a reference trajectory while avoiding collisions with objects in the plane. The robot is modeled by the following nonlinear ordinary differential equations (ODEs)

$$\begin{aligned}\dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= u\end{aligned} \tag{1}$$

where $x \in \mathbb{R}$ and $y \in \mathbb{R}$ are the Cartesian coordinates, $\theta \in [0, 2\pi)$ is the orientation of the robot with respect to a given frame and v, u are linear and angular velocity respectively. We are also given a reference trajectory for the robot to follow, denoted by x_d, y_d. Then, we define the position errors as $e_x = x - x_d$ and $e_y = y - y_d$. The coordinates of the object to be avoided are given by x_a, y_a and we can define

$$d_a = \sqrt{\left(\frac{x - x_a}{\alpha}\right)^2 + \left(\frac{y - y_a}{\beta}\right)^2} \tag{2}$$

for $\alpha, \beta > 0$. In this paper we will address the obstacle avoidance problem through the following potential function defined for $\alpha = \beta = 1$:

$$V_a = \left(\min\left\{0, \frac{d_a^2 - R^2}{d_a^2 - r^2}\right\}\right)^2 \tag{3}$$

defined in [15], where r is the radius of the avoidance region around the obstacle, and R is the radius of detection region around the obstacle, with $R > r > 0$. Thus, this function blows up whenever the robot approaches the avoidance region and is zero whenever the robot is outside the sensing region. To break the symmetry different shapes of the potential function, for example ellipsoids, can be obtained by choosing different values for the coefficients α, β. Upon taking the partial derivatives of V_a with respect to the x and y coordinates, we obtain

$$\frac{\partial V_a}{\partial x} = \begin{cases} 0, & \text{if } d_a \geq R \\ 4\frac{(R^2 - r^2)(d_a^2 - R^2)}{(d_a^2 - r^2)^3}(x - x_a), & \text{if } R > d_a > r \\ 0, & \text{if } d_a < r \end{cases} \tag{4}$$

and

$$\frac{\partial V_a}{\partial y} = \begin{cases} 0, & \text{if } d_a \geq R \\ 4\frac{(R^2 - r^2)(d_a^2 - R^2)}{(d_a^2 - r^2)^3}(y - y_a), & \text{if } R > d_a > r \\ 0, & \text{if } d_a < r \end{cases} \tag{5}$$

Let us define

$$E_x = e_x + \frac{\partial V_a}{\partial x}, \quad E_y = e_y + \frac{\partial V_a}{\partial y},$$

for $(E_x, E_y) \neq (0, 0)$, the desired orientation as

$$\theta_d = \text{Atan2}(E_y, E_x), \tag{6}$$

and the orientation error: $e_\theta = \theta - \theta_d$. Note that θ_d defines a desired direction of motion that depends on the reference trajectory, the robot position and on the obstacle to avoid. Some configurations might lead to singular directions. In order to avoid singular cases, we will assume throughout the paper that the reference trajectory has the following characteristics:

Assumption 1: The reference trajectory is smooth and satisfies:

$$|e_\theta| \leq \arccos(\delta_\theta) \tag{7}$$

for some $\delta_\theta \in (0, 1]$.

Assumption 2: The reference trajectory remains constant inside the collision region, i.e. $\dot{x}_d = \dot{y}_d = 0$, for $r \leq d_a < R$.

Assumption 3: Define $\hat{\theta}_d$ to be an estimate which entails some measurement error of

$$\hat{\theta}_d = \frac{E_x \hat{E}_y - \hat{E}_x E_y}{D^2} \tag{8}$$

where $D = \sqrt{E_x^2 + E_y^2}$. Then, we assume that

$$\left|\dot{\hat{\theta}}_d - \dot{\theta}_d\right| \leq \epsilon_\theta \tag{9}$$

for some small positive ϵ_θ. Note that most of the variables in $\hat{\theta}_d$ can be measured, in fact we have that $\left|\dot{\hat{\theta}}_d - \dot{\theta}_d\right| = \frac{E_x(\dot{\hat{E}}_y - \dot{E}_y) - E_y(\dot{\hat{E}}_x - \dot{E}_x)}{D^2}$, where the estimates can be chosen to as values $\dot{\hat{E}}_x = \frac{\hat{E}_x(t+T) - \hat{E}_x(t)}{T}$ and $\dot{\hat{E}}_y = \frac{\hat{E}_y(t+T) - \hat{E}_y(t)}{T}$, for some small T. Then, as E_x, E_y are smooth $a.e.$ we have that $(\dot{\hat{E}}_x - \dot{E}_x) \simeq (\hat{E}_x - E_x) \simeq o(T)$, and we can pick $\epsilon_\theta \simeq o(T)$. If we assume that $\dot{x}_d$, $\dot{y}_d$ can be measured then $\dot{\theta}_d$ can be exactly calculated and $\left|\dot{\hat{\theta}}_d - \dot{\theta}_d\right| = 0$.

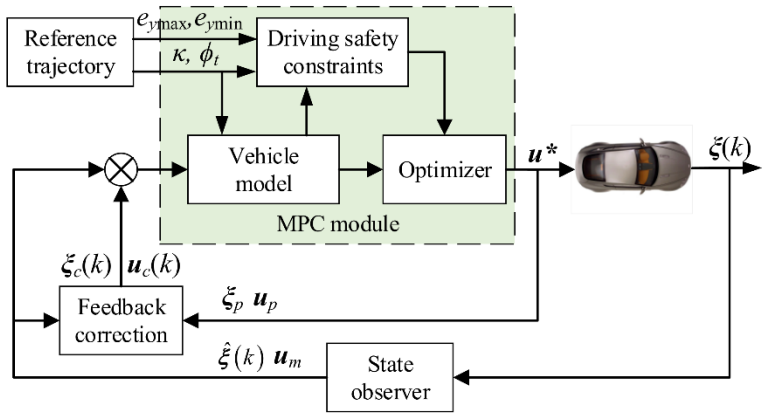
Remark 1: Assumption 1 on the reference trajectory implies the following two conditions:

1) Outside the collision region ($d_a \geq R$) and for $(e_x, e_y) \neq (0, 0)$ we have $\theta_d = \text{Atan2}(e_y, e_x)$. The reference trajectory is such that it does not initiate sharp turns of angle greater than $\arccos(\delta_\theta)$ with respect to the current orientation of the robot. Note that this condition is not too restrictive since the robot can reorient itself in place if the condition is not satisfied.
2) Inside the safety region ($r \leq d_a < R$) the reference trajectory x_d, y_d must be such that $|e_\theta| \leq \arccos(\delta_\theta)$. The way to achieve this is to consider a perturbed reference trajectory instead of the real one whenever (7) is not satisfied. Given a reference trajectory x_d, y_d that enters the safety region around an obstacle at the point x_a, y_a, and do not satisfy Assumption 1, we can replace the reference trajectory with the following perturbed version:

$$\begin{aligned}\bar{x}_d &= x_d - \text{sign}(x - x_a)\epsilon_x \\ \bar{y}_d &= y_d - \text{sign}(y - y_a)\epsilon_y\end{aligned} \tag{10}$$

where $\epsilon_x \neq \epsilon_y$ are some small perturbation values, and $\text{sign}(y - y_a)$, $\text{sign}(x - x_a)$ define the direction opposite to the obstacle, i.e. we want to perturb the trajectory in the direction opposite to the obstacle. This condition

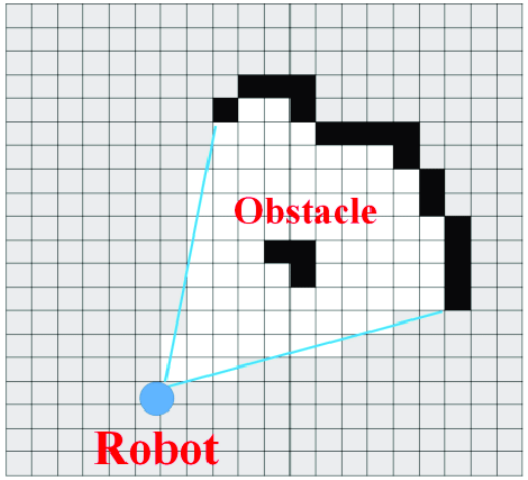
Remote Formation Control and Collision Avoidance for Multi-Agent Nonholonomic Systems



충돌 회피 알고리즘 – 경로 계획

I. Grid Map based Planning

- 장점
  - ✓ 점유 격자 맵 (Occupancy grid map) 기반
  - ✓ 다른 경로 계획 알고리즘과 조합하기 용이함
- 단점
  - ✓ 장애물의 크기에 따른 격자 크기 설정 문제
  - ✓ Free Space 주행(Ex. 로봇 청소기)의 경우 Random Point Map 의 정밀도에 따라 장애물 회피 궤적이 과도하게 발생할 수 있음



II. Nearness Diagram navigation

- 장점
  - ✓ 상황에 따른 적정 알고리즘을 설정 가능함
  - ✓ 각 상황에 대한 정교한 반응이 가능
- 단점
  - ✓ 수많은 상황을 고려해야함
  - ✓ 각 상황에 적합한 알고리즘을 전부 생성 해야함
  - ✓ 복잡한 상황이 연속적으로 발생할 경우 발산할 수도 있음

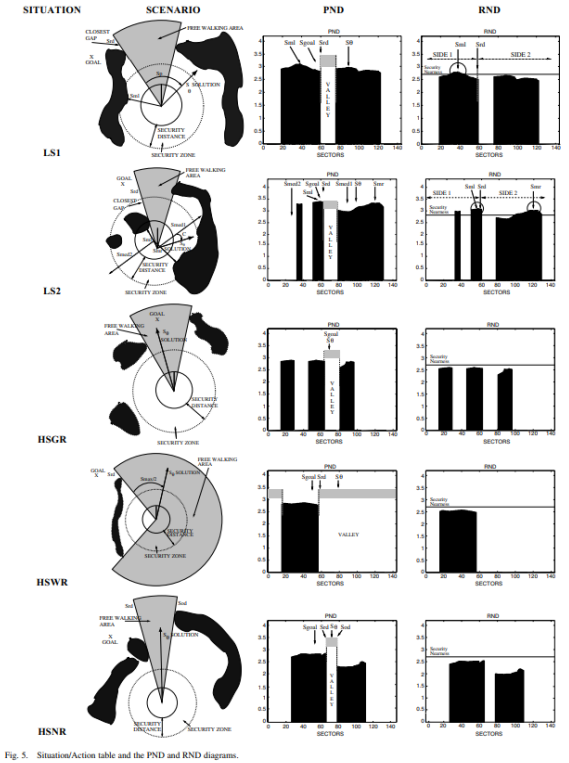


Fig. 5. Situation/Action table and the PND and RND diagrams.

충돌 회피 알고리즘 – 경로 계획

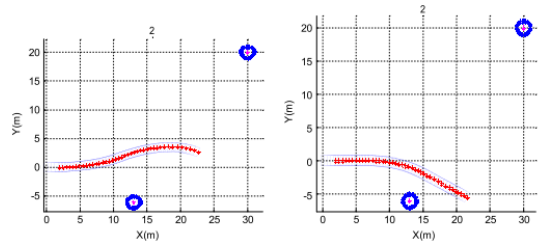
I. A* (A – Star algorithm)

- 장점
  - ✓ 가장 많이 사용됨
  - ✓ Grid Map, Graph Map 등에서 빠른 길 찾기 용이함
- 단점
  - ✓ Grid, Graph와 같은 점 – 링크 기반의 데이터 구조에서 큰 효력을 가짐
  - ✓ 존재하는 포인트에 대해서만 시작, 종료가 가능함
  - ✓ 단순 A* 알고리즘은 움직임이 너무 sharp하기 때문에 플랫폼에 따라서 부적합한 움직임을 만들어 냄

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

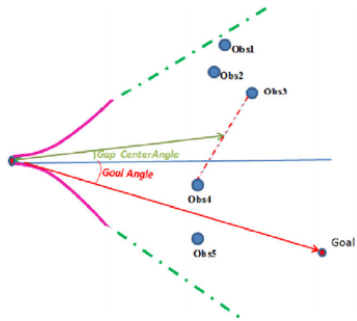
II. Follow the Gap

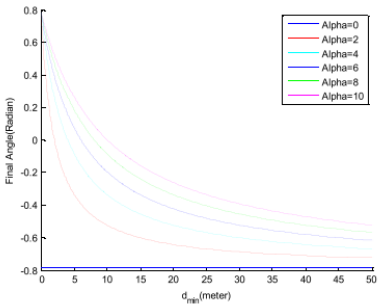
- 장점
  - ✓ 간단한 알고리즘
  - ✓ Smooth한 이동 궤적
  - ✓ 전체 경로를 새로 탐색하지 않아도 됨
- 단점
  - ✓ 복잡한 주행 조건에서는 사용하기 힘듦
  - ✓ 다수의 동적 장애물에 대한 경로 대응이 쉽지 않음
  - ✓ 센서의 정확도 의존성이 큼



(a) Result of ϕ_{gap_c} (FGM). (b) Result of $\phi_{gap\_c\_basic}$ (FGM-basic).

Fig. 9. Comparison of ϕ_{gap_c} and $\phi_{gap\_c\_basic}$ gap angle calculation.





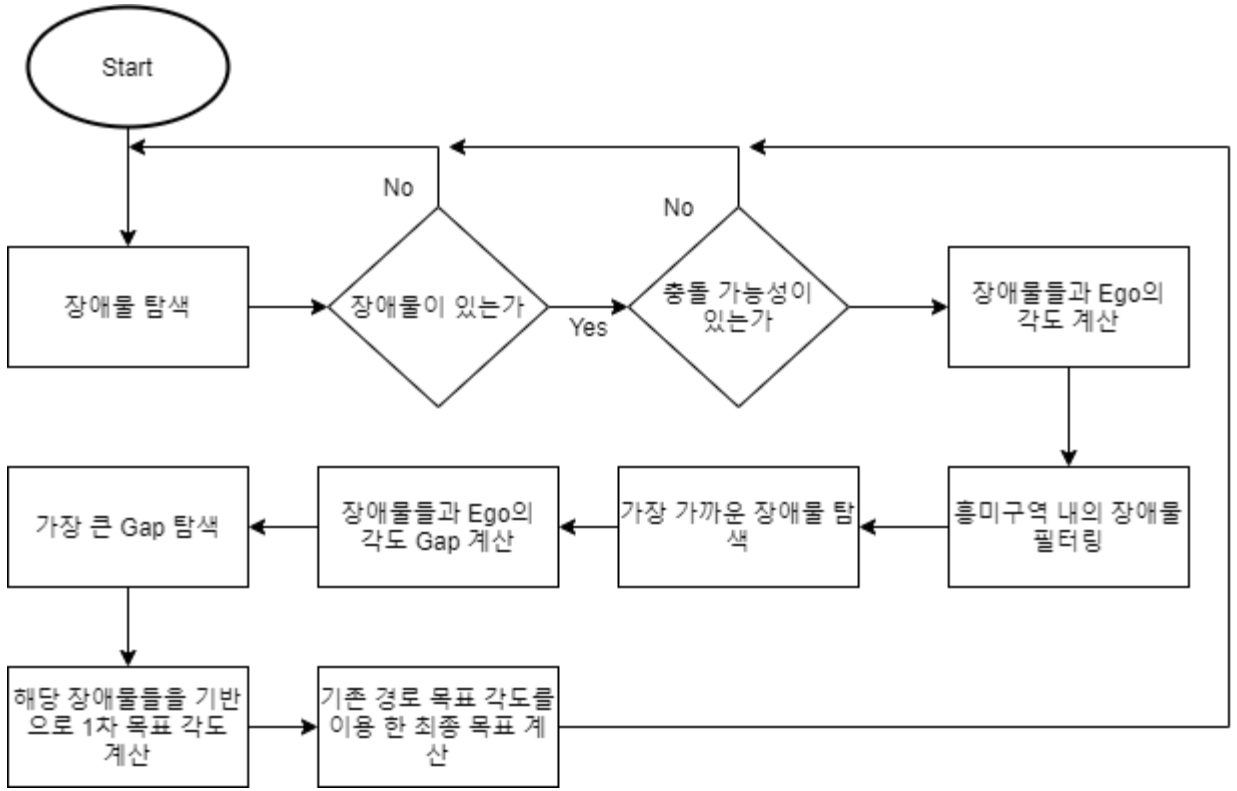
Follow the Gap

I. Follow the Gap의 기본 흐름

- 인지 혹은 GT 데이터를 이용하여 장애물을 탐색
- 각 장애물들과 Ego 차량 간의 각도 계산
- 장애물이 존재할 경우 관심 영역 (Interest area) 내에 있는 장애물만 필터링
- 장애물들 과의 최대 각도와 최소각도 계산
- 각도 간격 (Gap)을 계산하여 최대 구역을 탐색
- 1차 목표 각도를 산출
- 가장 가까운 장애물과의 거리를 반영하여 최종 제어 입력 산출

II. 응용 설계

- 장애물 종류에 따른 최대 충돌 거리 모델 변경
- 계산된 횡 방향 제어 입력을 조건에 따라 사용 여부 판단
- 현재 속도에 따른 관심 영역 범위 조절



Sezer, Volkan, and Metin Gokasan. "**A novel obstacle avoidance algorithm:"Follow the Gap Method".**" *Robotics and Autonomous Systems* 60.9 (2012): 1123-1134.

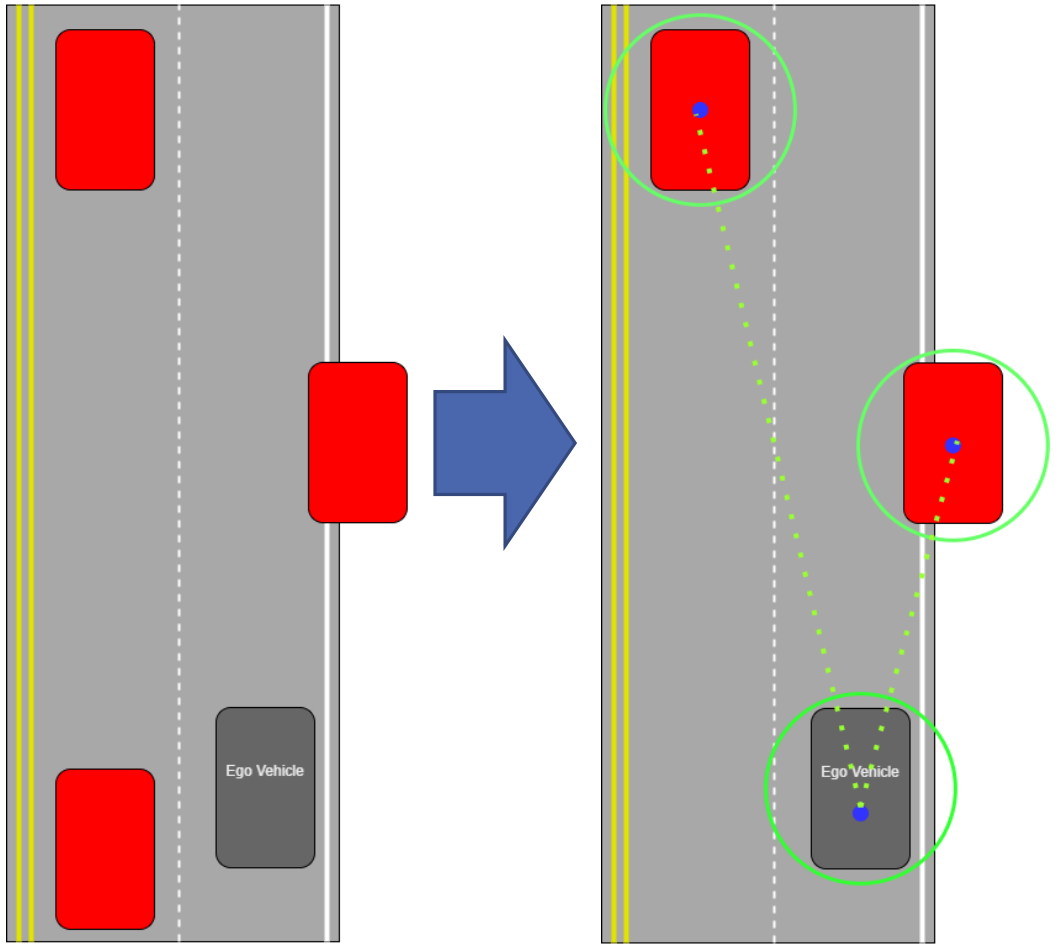
Follow the Gap – basic

Make guidance angle

- 주행 경로와 차선 이동에 해당되는 기준 범위 각도를 선정
- $\phi_{lim} = [\phi_{limR}, \phi_{limL}]$

Calculate obstacle angles

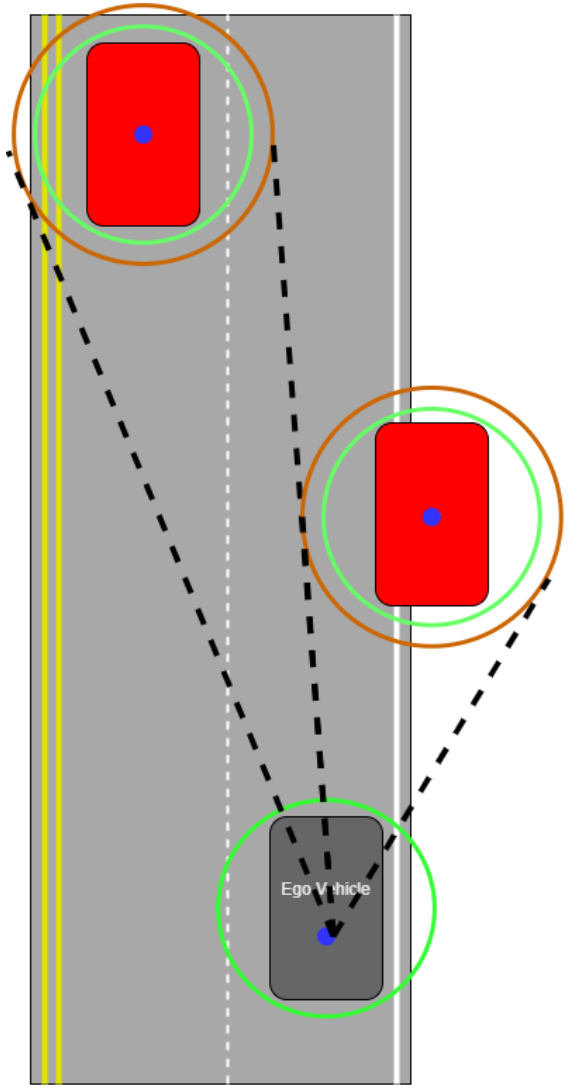
- 인지된 모든 물체들에 대하여 각도를 계산함.
  - ✓ 각 차량 (붉은색 상자)의 기준점과 나의 기준점의 거리를 계산함.
  - ✓ 각도의 기준은?
    - 나의 Center Of Mass
    - 감지된 물체의 예상되는 중심점
    - 기준 범위 각도 이상의 물체는 고려하지 않음
- $\phi_{Obj} = tan^{-1}\left(\frac{(Y_{Obj} - Y_{Ego})}{(X_{Obj} - X_{Ego})}\right)$



Follow the Gap – basic

- **Sort objects**
  - 가장 좌측의 물체를 첫번째로 (CCW 방향이 양의 방향)
  - 각도 순서대로 물체들을 정렬

- **Calculate left/right angle**
  - 각 물체의 최 좌측, 최 우측 각도를 계산
  - $dist_{i,side} = \sqrt{(x_i - x_{ego})^2 + (y_i - y_{ego})^2 - (r_i + dist_{margin})^2}$
  - $\Delta\phi_i = tan^{-1}\left(\frac{r_i}{dist_{i,side}}\right)$
  - $\phi_{i,left} = \phi_i + \Delta\phi_i \quad \phi_{i,right} = \phi_i - \Delta\phi_i$
- **Calculate gap**
  - 각 물체들 간 최대 충돌 거리를 고려하기 위한 각 물체의 사이드 각도 차이를 계산
  - $Gap = \{\Delta\phi_{gap} | \Delta\phi_{i,gap} = \phi_{i+1,left} - \phi_{i,right}\}$
- **Find maximum gap**
  - Gap 에서 가장 큰 값을 선택
  - 선택된 Gap 에서 유도된 물체 2개를 선택



Follow the Gap – basic

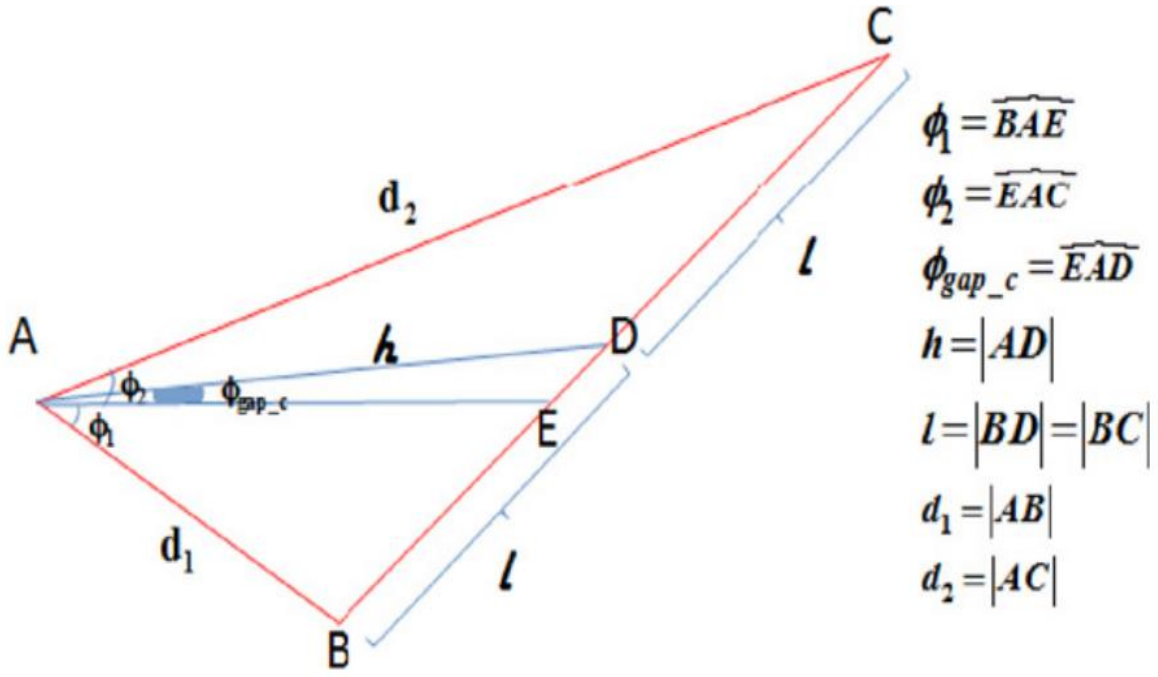
- **Calculate gap center angle**
  - 선택한 물체를 우측 그림의 B,C로 설정
  - 코사인 룰에 의해서 l 을 구할 수 있음
  - $(2l)^2 = d_1^2 + d_2^2 - 2\,d_1 d_2\,cos(\phi_1 + \phi_2)$
  - 위 식을 이용하여 목표 점 D를 향하는 각도는 다음과 같이 설정됨

$$\phi_{gap\_c} = cos^{-1}\left(\frac{d_1 + d_2 cos(\phi_1 + \phi_2)}{\sqrt{d_1^2 + d_2^2 + 2\,d_1 d_2 cos(\phi_1 + \phi_2)}}\right) - \phi_1$$

  - 기존 목표 방향 E에 대한 term을 추가하면 다음과 같이 최종 목표 방향이 설정됨

$$\phi_{final} = \frac{\frac{\alpha}{d_{min}}\phi_{gap\_c} + \beta\phi_E}{\frac{\alpha}{d_{min}} + \beta}$$

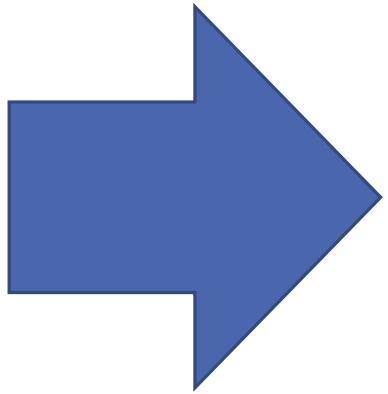
  - α와 β는 제어 상수 (설계자가 정함)



A: Ego
B, C: 선택된 물체
D: 차량 주행 방향
E: 원래 주행 방향

Follow the Gap – basic

```
1   Header header
2   int32 num_of_objects
3   int16[] object_type
4
5   float64[] pose_x
6   float64[] pose_y
7   float64[] pose_z
8
9   float64[] heading
10
11  float64[] size_x
12  float64[] size_y
13  float64[] size_z
14
15  float64[] velocity
```



```
1   # frame and time stamp
2   Header header
3
4   # should we do collision avoidance
5   bool do_ca
6
7   float32 ca_const_alpha
8   float32 ca_const_beta
9   float32 ca_distance
10
11  float32 phi_gap
```

Follow the Gap – basic

```python
class ObjInfo(object):
    def __init__(self, pos, heading, size, objType):
        self.posX = pos[0]
        self.posY = pos[1]
        self.heading = heading
        self.sizeX = size[0]
        self.sizeY = size[1]
        self.objType = objType
        self.maxCircleRadius = np.sqrt(max(size)**2)/2 + MarginDist

        self.angleFromEgo = 0.0
        self.distFromEgo = 0.0

        self.marginAngleLeft = 0.0
        self.marginAngleRight = 0.0
        self.marginDist = 0.0
```

```python
def callbackObjInfo(self, datas):
    self.len_obj = datas.num_of_objects
    if self.len_obj > 0:
        posX = datas.pose_x
        posY = datas.pose_y
        heading = datas.heading
        sizeX = datas.size_x
        sizeY = datas.size_y
        objType = datas.object_type
        self.list_obj = []
        for i in range(0,self.len_obj):
            self.list_obj.append(
                ObjInfo([posX[i], posY[i]], heading[i],
                        [sizeX[i], sizeY[i]], objType[i]))
    else:
        self.list_obj = []
```

Follow the Gap – basic

```python
def filtering(self):
    tmpNpArr = np.array(self.list_obj, dtype=object)
    self.list_obj = []
    tmpFilterList = []
    maxIdx = 0
    for obj in tmpNpArr:
        obj.angleFromEgo = np.arctan2((obj.posY - self.egoPosY), (obj.posX - self.egoPosX)) - self.egoHeadAngle
        obj.distFromEgo = np.linalg.norm([(obj.posX - self.egoPosX), (obj.posY - self.egoPosY)], ord=2)
        tmpFilterList.append(obj.angleFromEgo)
    tmpFilterArr = np.asarray(tmpFilterList)
    tmpMask = np.logical_and((tmpFilterArr < self.guidAngleLeft), (tmpFilterArr > self.guidAngleRight))
    filteredDatas = tmpNpArr[tmpMask]
    filteredAngles = tmpFilterArr[tmpMask]
    sortedDatas = []
    if(len(filteredDatas) > 0):
        sortedDatas = sorted(filteredDatas, key=lambda x: x.angleFromEgo, reverse=True)
    return sortedDatas
```

Follow the Gap – basic

```python
def calcGap(self, objs, length):
    angleList = [[0, self.guidAngleLeft]]
    gapList = []
    for i in range(0, length):
        distDatas = [(objs[i].posX - self.egoPosX), (objs[i].posY - self.egoPosY), (objs[i].maxCircleRadius + MarginDist)]
        tmpSquare = distDatas[0]**2 + distDatas[1]**2 - distDatas[2]**2
        if tmpSquare <= 0:
            tmpSquare = distDatas[0]**2 + distDatas[1]**2
        distSide = np.sqrt(tmpSquare)
        objs[i].marginDist = distSide
        deltaAngle = np.arctan2(objs[i].maxCircleRadius, distSide)
        tmpLeftAngle = objs[i].angleFromEgo + deltaAngle
        tmpRightAngle = objs[i].angleFromEgo - deltaAngle
        objs[i].marginAngleLeft = tmpLeftAngle
        objs[i].marginAngleRight = tmpRightAngle
        angleList.append([tmpLeftAngle, tmpRightAngle])
    angleList.append([self.guidAngleRight, 0])
    for i in range(1, len(angleList)):
        tmpDeltaAngleLeft = (angleList[i-1][1] - angleList[i][0])
        gapList.append(tmpDeltaAngleLeft)
    npGapList = np.asarray(gapList)
    maxIdx = np.argmax(npGapList)
    return (maxIdx, npGapList)
```

Follow the Gap – basic

```python
def process(self):
    tmpResultMsg = CollisionAvoidance()
    if self.len_obj > 0:
        focusObjs = self.filtering()
        lengthObjs = len(focusObjs)
        if lengthObjs < 1:
            return tmpResultMsg
        findNearObjs = sorted(focusObjs, key=lambda x: x.distFromEgo, reverse=False)
        maxIdx, gapList = self.calcGap(focusObjs, lengthObjs)
        if (maxIdx == 0):
            maxIdx = 1
        elif(maxIdx == (len(gapList))):
            return tmpResultMsg
        tmpIdx = maxIdx - 1
        tmpDist1 = np.sqrt(focusObjs[tmpIdx].maxCircleRadius**2 + focusObjs[tmpIdx].distFromEgo**2)
        tmpDist2 = np.sqrt(focusObjs[tmpIdx-1].maxCircleRadius**2 + focusObjs[tmpIdx-1].distFromEgo**2)
        sumAngle = focusObjs[tmpIdx].marginAngleLeft + focusObjs[tmpIdx-1].marginAngleRight
        tmpNum = tmpDist1 + tmpDist2 * np.cos(sumAngle)
        tmpDen = np.sqrt( tmpDist1**2 + tmpDist2**2 + (2.0 * tmpDist1 * tmpDist2 * np.cos(sumAngle)) )
        angleGapC = np.arccos(tmpNum/tmpDen) - (focusObjs[tmpIdx].marginAngleLeft)
        tmpResultMsg.do_ca = True
        tmpResultMsg.phi_gap = -angleGapC
        tmpResultMsg.ca_distance = findNearObjs[0].distFromEgo
        tmpResultMsg.ca_const_alpha = GainAlpha
        tmpResultMsg.ca_const_beta = GainBeta

    return tmpResultMsg
```

Follow the Gap – advance

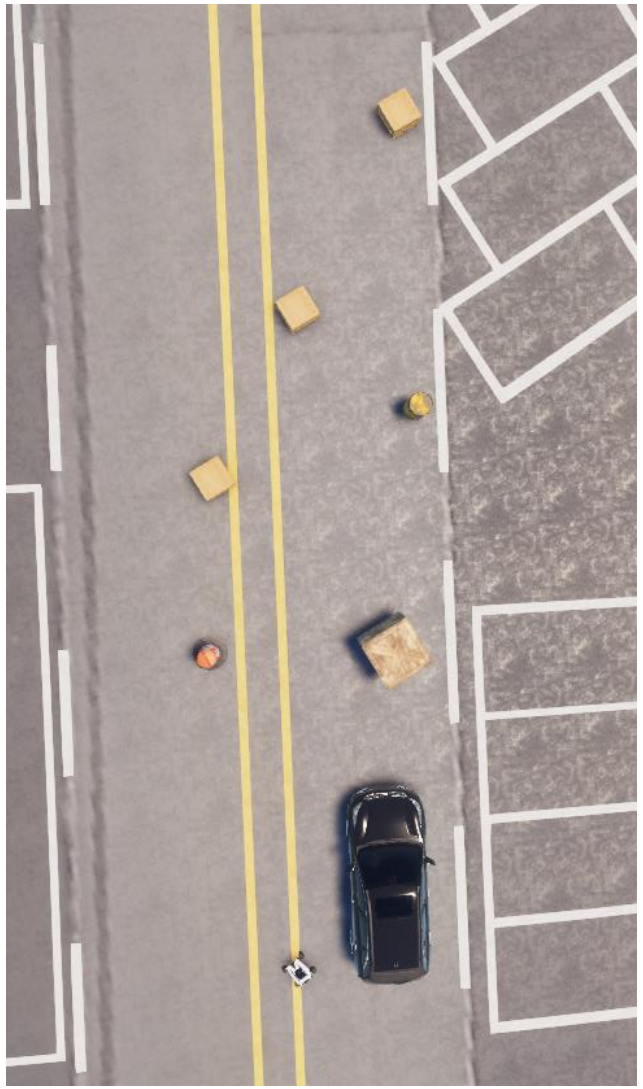
- **장애물 종류에 따른 최대 충돌 거리 모델 변경**
  - **장애물의 모양은?**

Follow the Gap – advance

I. 장애물 종류에 따른 최대 충돌 거리 모델 변경

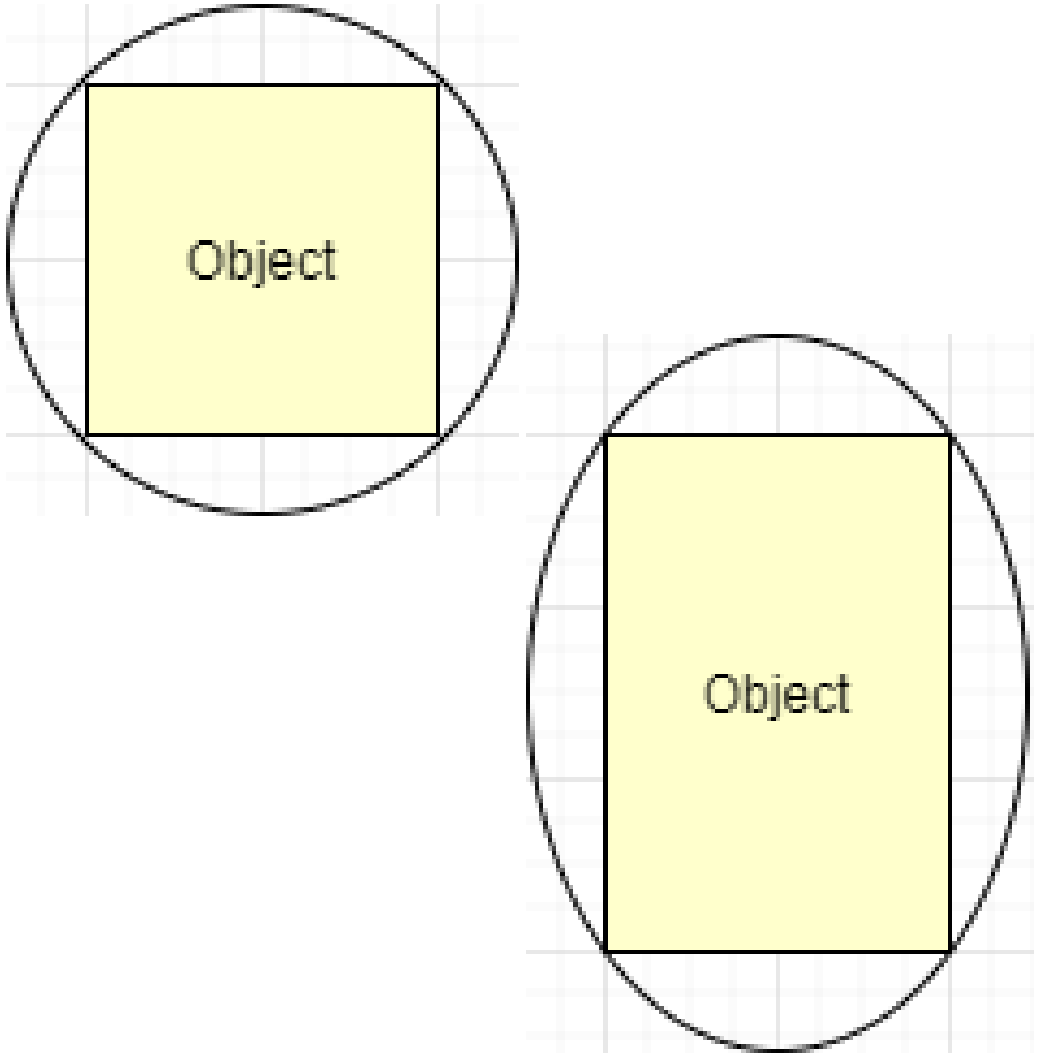
- 왜 원형으로 물체를 모사하는가
  - ✓ 다각형을 코드로 모사하기 힘듦
    - ❑ 물체 하나를 모사하기 위해 많은 처리가 필요
    - ❑ 이에 따른 많은 연산이 필요
  - ✓ 수학적으로 smooth하지 않음
    - ❑ 많은 오차가 발생
    - ❑ 부자연스러운 로봇의 움직임이 발생

- 원형 모델의 이점
  - ✓ 연속적인 처리가 가능
    - ❑ 모서리와 같은 특정 케이스에 대한 예외처리가 필요하지 않음
  - ✓ 계산 오차가 감소
  - ✓ 여러 모형의 물체를 단순화 시킬 수 있음
  - ✓ 계산이 용이함
    - ❑ 알고리즘 수식에서 고려할 변수가 줄어듦



Follow the Gap – advance

- 장애물 종류에 따른 최대 충돌 거리 모델 변경
  - 원 모형
    - ✓ $x^2 + y^2 = r^2$
    - ✓ 정사각형 모델은 원 모형 안에 들어갈 수 있음
      - $r = \frac{d_{diag}}{2}$
  - 타원 모형
    - ✓ $\frac{x^2}{\gamma} + \frac{y^2}{\zeta} = 1$
    - ✓ 직사각형 모델은 타원 모형 안에 들어갈 수 있음
      - $\gamma = \frac{d_w}{\sqrt{2}} \quad \zeta = \frac{d_h}{\sqrt{2}}$



Follow the Gap – advance

❖ 장애물 종류에 따른 최대 충돌 거리 모델 변경

- **타원과 원 사이의 거리는?**
  - ✓ 원의 반지름은 이미 알고있다
    -> 타원과 점과의 거리를 계산하여 반지름을 뺀다
  - ✓ 거리는 점과 점 사이의 거리를 계산
    -> 타원에서 점과 가장 가까운 점을 찾는다
    -> 타원의 축폐선 (ellipse evolute)를 이용

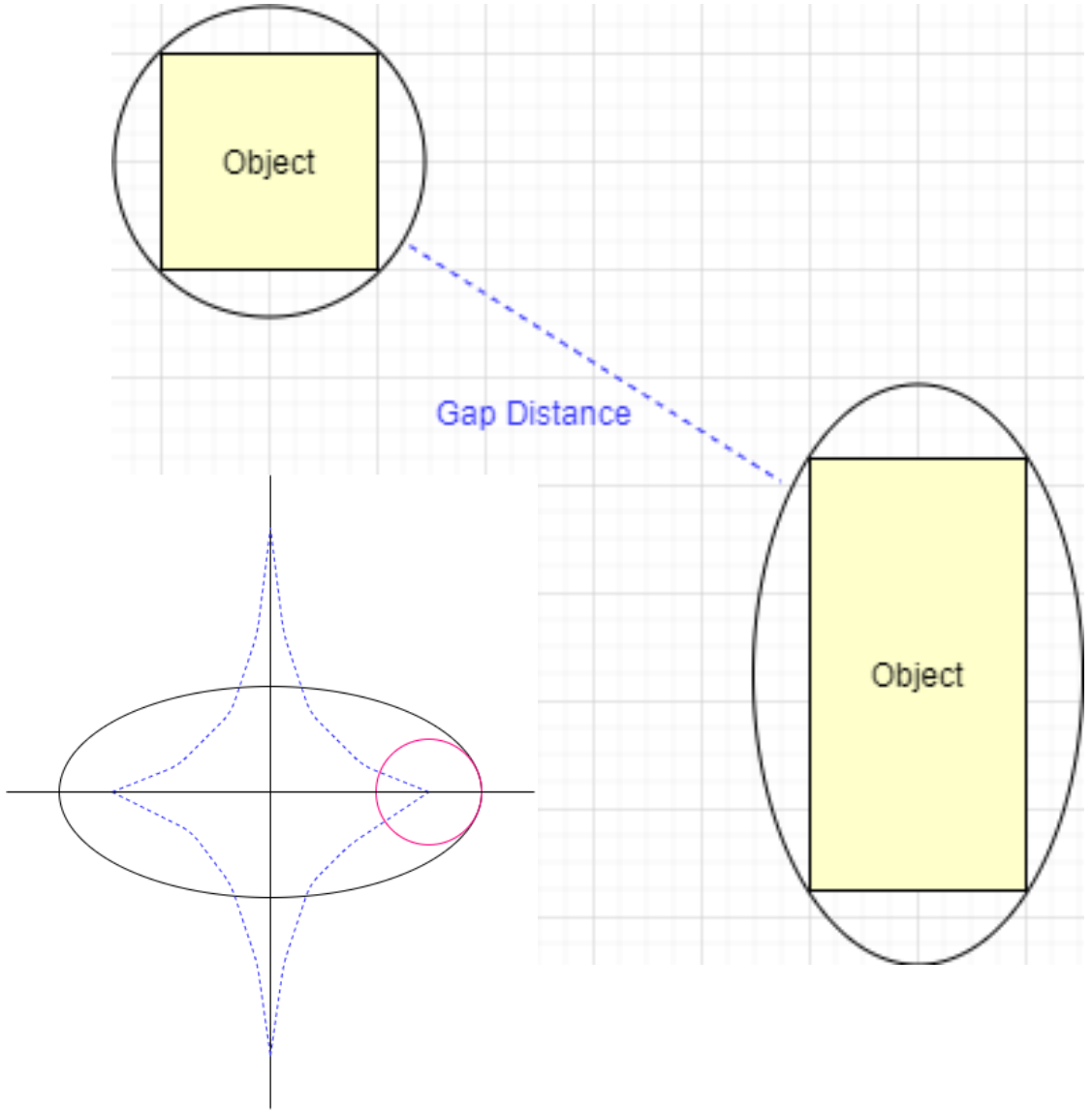
$$R(\theta) := (\gamma\, cos(\theta),\ \zeta\, sin(\theta))$$

$$x_{evol} = \frac{\gamma^2 - \zeta^2}{\gamma} cos(\theta)^3 \quad y_{evol} = \frac{\zeta^2 - \gamma^2}{\zeta} sin(\theta)^3$$

$$\Delta_{ev} = d_{(el,ev)}\, sin^{-1}\left(\frac{x_{(el,\theta)} y_{(pt,ev)} - y_{(el,\theta)} x_{(pt,ev)}}{d_{(el,ev)}\, d_{(pt,ev)}}\right)$$

$$\Delta_\theta = \frac{\Delta_{ev}}{\sqrt{\gamma^2 + \zeta^2 - x^2_{(el,\theta)} - y^2_{(el,\theta)}}}$$

  - ✓ 계산 값은 소수점 길이가 길기 때문에 유효 숫자를 정함
    -> 일정 수준의 오차율을 감수하는 대신 속도가 빠름



Follow the Gap – advance

```python
import math

def EllipseDistance(semi_major, semi_minor, point):
    px = abs(point[0])
    py = abs(point[1])

    t = 0

    a = semi_major
    b = semi_minor

    for i in range(0, 4):
        print(i)
        x = a * math.cos(t)
        y = b * math.sin(t)
        print(x,y)

        ex = (a**2 - b**2) * math.cos(t)**3 / a
        ey = (b**2 - a**2) * math.sin(t)**3 / b

        rx = x - ex
        ry = y - ey

        qx = px - ex
        qy = py - ey

        r = math.hypot(ry, rx)
        q = math.hypot(qy, qx)

        delta_c = r * math.asin((rx*qy - ry*qx)/(r*q))
        delta_t = delta_c / math.sqrt(a**2 + b**2 - x**2 - y**2)

        t += delta_t
        t = min(math.pi/2, max(0, t))
        print((x, y))

    return (round(math.copysign(x, point[0]), 6), round(math.copysign(y, point[1]), 6))
```

Follow the Gap – advance (homework)

❖ 현재 속도에 따른 관심 영역 범위 조절

- 속도가 높을 수록 횡가속도가 높아짐
- 높은 횡가속도는 주행 불능 혹은 사고를 야기함
  - ✓ Ex. Understeer, Oversteer, Rollover
- 범위 조절 알고리즘 설계
  - ✓ 여러 각도를 대입하여 주행 경향성 분석
  - ✓ 경향에 따른 선형 / 비선형 수식 설계 및 대입

❖ 계산된 횡 방향 제어 입력을 조건에 따라 사용 여부 판단

- 계산된 횡 방향 제어 입력을 사용하지 않을 경우
  - ✓ 목표 속도 조절 (저속 / 정지)
  - ✓ Adaptive Cruise Control
  - ✓ Etc.
- 조건 설계
  - 이동에 영향을 미치는 상황 분석
  - 사고 발생 요소 분석