TIPE

September 5, 2020

1 Description du sujet

1.1 Problématiques

- Peut-on trouver un chemin idéal entre deux points d'un ensemble non-convexe ?
- Comment se déplacer efficacement ?

1.2 Liens avec le thème

Lien avec les enjeux sociétaux :
- Construction : Tracer des routes sans détruire des zones protégées tout en optimisant le temps de trajet (Environement, ~~Energie)
- Voitures autonomes : Utiliser la meilleur trajectoire possible (Securité, ~~Energie)

1.3 Contacts

Contrairement à la lévitation acoustique, il est possible d'avoir des contacts dans ce domaine et nous avons déjà travaillé avec la plupart d'entre eux :

- Olivier Ly : Bordeaux

- Grégoire Passault : Bordeaux

Et plus globalement les chercheurs du groupe de la Rhoban à Bordeaux

1.4 Expérience

Une expérience serait d'implémenter nos algorithmes sur des robots qui participe à la ligue SSL de la Robocup (Vidéo, Description.

Cela pourait être réalisé avec une équipe naissante à Pau ou l'équipe de la Rhoban/Namec à Bordeaux.

2 Recherches

2.1 Définitions

2.1.1 Pathfinding

> En robotique mobile, planifier un déplacement devient encore plus complexe. En effet, il s'agit de se déplacer dans un environnement réel. Tout d'abord, le robot ne dispose que d'une estimation de sa position, car ses capteurs ne sont pas parfaits. De la même façon, il doit se déplacer au moyen d'effecteurs qui ne peuvent l'amener là où il décide qu'avec une certaine précision.
>
> Afin de prendre en compte ces incertitudes, il est nécessaire de passer à des modèles mathématiques probabilistes (comme les MDP et les POMDP).
>
> De plus, si on ajoute dans l'environnement des humains ou des animaux, il faut prévoir comment ces entités vont se déplacer afin de les éviter.

Wikipedia

2.2 Algorithmes de pathfinding

- Algorithme de Dijkstra : Wikipedia
- Algorithme A étoile : Wikipedia
- Rapidly-exploring random tree : Wikipedia

On peut y faire une étude théorique sur les ensembles non-convexes

2.3 Prises de Decisions

Modèles probabilistes comme le Processus de décision markovien

2.4 Modèle mathématique

2.4.1 Ensembles convexes

> On utilisera le terme *non convexe* à la place du terme *concave* pour eviter les confusions entre ensembles et fonctions :
>
> « An often seen confusion is a "concave set". Concave and convex functions designate certain classes of functions, not of sets, whereas a convex set designates a certain class of sets, and not a class of functions. A "concave set" confuses sets with functions. » -- Akira Takayama

Définition 1 : On a équivalence entre les trois proposition suivantes :
1- C est convexe
2- $\forall x, y \in C, [x, y] \in C$
3- $\forall x, y \in C, \forall t \in [0,1], tx + (1-t)y \in C$

Autrement dit, On peut relier deux points d'un ensemble sans en sortir si et seulement si il est convexe.

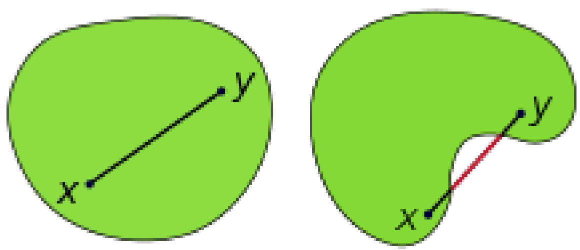
Proposition 1 : L'intersection de deux convexes est elle-même convexe

Demonstration :

Soit A et B deux ensembles convexes.
Montrons que $A \cap B$ est lui aussi convexe - Si $A \cap B = \{\emptyset\}$ alors c'est fini - Sinon :

Soit $x, y \in A \cap B$ et $t \in [0,1]$, il vient que x et y sont dans A mais aussi dans B.
Or A et B sont convexes,
Par définition, $tx + (1-t)y \in A$ et $tx + (1-t)y \in B$.
Donc $tx + (1-t)y \in A \cap B$.
Donc $A \cap B$ est convexe.
L'intersection de deux ensembles convexes est aussi convexe.

Exemple 1 : Ensembles quelconques :



Exemple 2 : Notons C un ensemble definie par $\{(x,y) \in \mathbb{R}^2, 1 \leq x^2 + y^2 \leq 3\}$

Preuve :

On utilise la propriété : $\forall x, y \in C, \forall t \in [0,1], tx + (1-t)y \in C$

On choisit $x = (-\frac{1}{2}, 0)$, $y = (\frac{1}{2}, 0)$ et $t = \frac{1}{2}$
On a bien $x, y \in C$ et $t \in [0,1]$

Or $tx + (1+t)y = \frac{1}{2}(-\frac{1}{2}, 0) + (1 - \frac{1}{2})(\frac{1}{2}, 0) = (-\frac{1}{4}, 0) + (\frac{1}{4}, 0) = (0,0) \notin C$

Donc C n'est pas convexe.

Exemple 3 : Notons C un ensemble definie par $\{(x,y) \in \mathbb{R}^2, x + y \geq 0\}$

Preuve :

On utilise la propriété : $\forall x, y \in C, \forall t \in [0,1], tx + (1-t)y \in C$
Soit $a, b \in C$, $t \in [0,1]$ et $x_1, y_1, x_2, y_2 \in \mathbb{R}$ tq $a = (x_1, y_1)$ et $b = (x_2, y_2)$

On a :

$ta + (1-t)b = t(x_1, y_1) + (1-t)(x_2, y_2) = (tx_1, ty_1) + ((1-t)x_2, (1-t)y_2) = (tx_1 + x_2 - tx_2, ty_1 + y_2 - ty_2)$

D'où

$ta + (1-t)b \in C$
$\Leftrightarrow tx_1 + x_1 - tx_2 + ty_1 + y_2 - ty_2 \geq 0$
$\Leftrightarrow t(x_1 - x_2 + y_1 - y_2) + x_2 + y_2 \geq 0$
$\Leftrightarrow t(x_1 + y_1 - (x_2 + y_2)) + x_2 + y_2 \geq 0$
$\Leftrightarrow t(x_1 + y_1) - t(x_2 + y_2) + x_2 + y_2 \geq 0$

Or

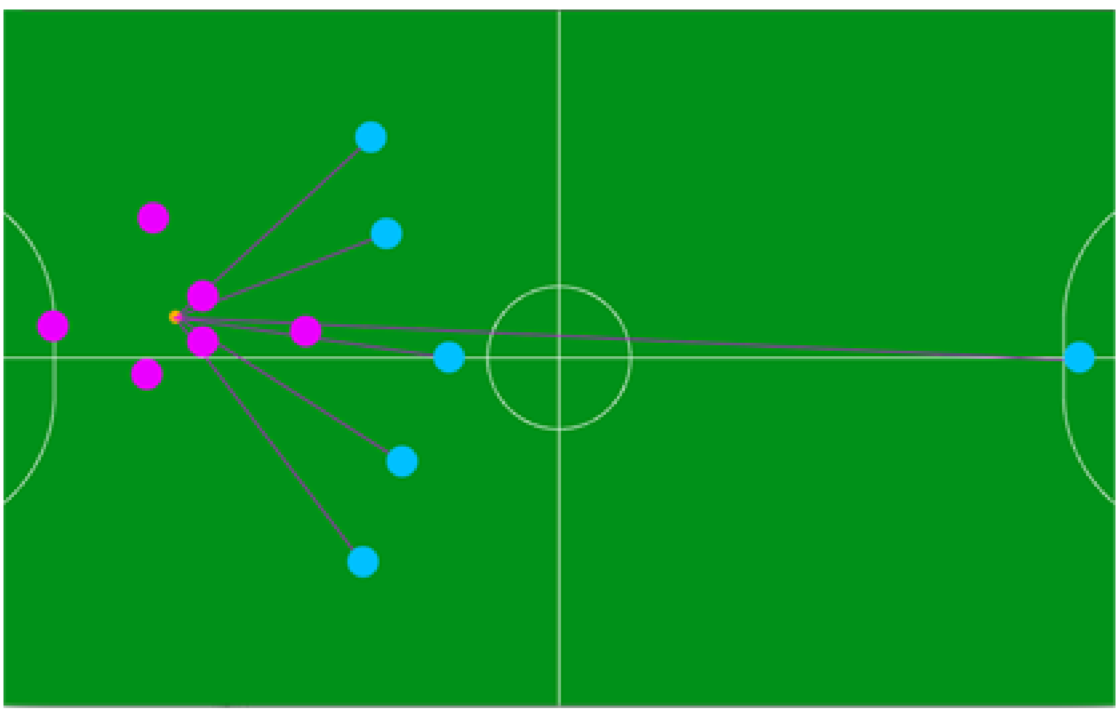
$t \in [0,1], (x_2, y_2) \in C$ donc $-t(x_2 + y_2) + x_2 + y_2 \geq 0$
et $t > 0, (x_1, y_1) \in C$ donc $t(x_1 + y_1) \geq 0$

On a donc : $t(x_1 + y_1) - t(x_2 + y_2) + x_2 + y_2 \geq 0$

Par équivalence, $ta + (1-t)b \in C$ donc C est convexe.

2.4.2 Liens avec l'experience

Nous pouvons ainsi modéliser notre terrain par un ensemble non convexe : avec les points bleus représentant les robots aliées, les points magentas représentant les robots adversaire et le point orange représentant l'objectif, la balle.



Notre ensemble est donc le terrain privé des robots présents.
Nous avons donc plusieurs strategies abordables, par exemple l'utilisation d'algorithmes de path-planning comme le RRT ou l'utilisation direct d'un algotithme "maison" qui utiliserai les ensembles localements convexes

2.5 Modèle numérique

2.5.1 Modélisation d'un ensemble

On choisit de représenter un ensemble E par une fonction Python qui renvoie si un élément X est dans E

Par exemple, pour l'ensemble $A = \{(x, y) \in \mathbb{R}^2, 1 \leq x^2 + y^2 \leq 3\}$, on a :

[1]:
```
A = lambda x,y: (1<= x**2+y**2) and (x**2+y**2 <= 3)
```

On peut ainsi implémenter une fonction qui indique plus simplement si un élément est dans un ensemble. On a :

[2]:
```
def EstPointDe(E,X):
    """
    E est un ensemble et X un élément de cette ensemble
    """
    x,y = X
    return E(x,y)
```

On a donc :

[3]:
```
X1 = (1,0)

EstPointDe(A,X1)
```

[3]:
```
True
```

2.5.2 Fonctions élémentaires

On a maitenant besoin d'implémenter les fontions élémentaires pour manipuler les couples :

[4]:
```
def addCouple(X,Y):
    """
    X et Y sont deux couples
    """
    x1,y1=X
    x2,y2=Y
    return (x1+x2,y1+y2)

def multCouple(X,k):
    """
    X est un couple et   est un scalaire
    """
    x,y=X
    return (x*k,y*k)
```

2.5.3 Chemins convexes

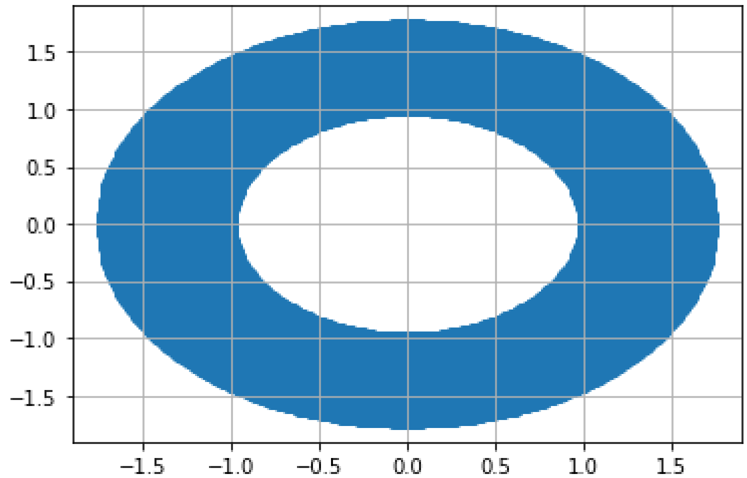
On peut donc maintenant implémenter une fonction qui verifie si un chemin est convexe

[5]:
```
def estCheminConvexe(E,X,Y,pas=0.01):
    """
    E est un ensemble, X,Y sont deux éléments de E
    """
    if EstPointDe(E,X) and EstPointDe(E,Y):
        t=0
        res=True
        while t<1:
            if not EstPointDe(E,addCouple(multCouple(X,t) ,␣
↪multCouple(Y,(1-t)))):
                res=False
            t+=pas
        return res
    else:
        return False
```

On peut donc maintenant faire une fonction qui trace l'ensemble et les chemins vers un point Y ...

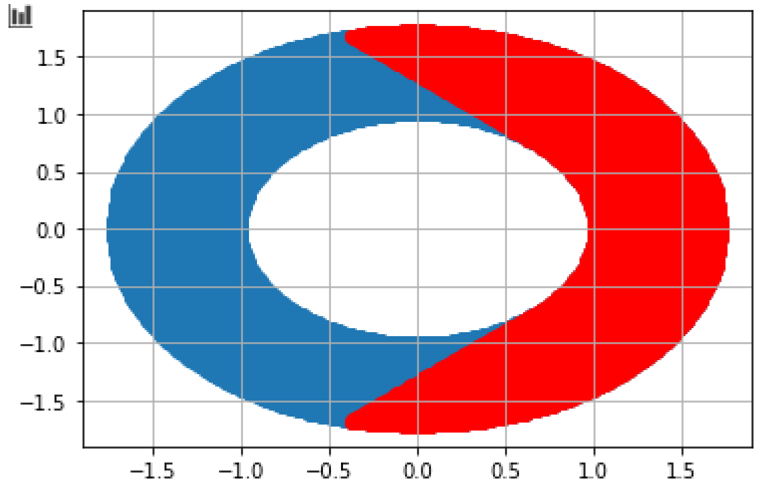
[6]:
```
import matplotlib.pyplot as plt

def traceEnsemble(E,deb,fin,pas=0.01):
    """
    E est un ensemble, deb est le coin supérieur gauche de la fenêtre, fin est␣
↪le coin inférieur droit de la fenêtre
    """
    x1,y1=deb
    x2,y2=fin
    resX = []
    resY = []
    x,y=x1,y1
    while x<x2:
        while y>y2:
            if EstPointDe(E,(x,y)):
                resX.append((x))
                resY.append((y))
            y-=pas
        y=y1
        x+=pas
    plt.scatter(resY,resX)
    plt.grid()
    plt.show()
```

Avec A on obtient :



[7]:
```python
def traceEnsembleEtChemin(E,deb,fin,Y,pas=0.01):
    """
    E est un ensemble, deb est le coin supérieur gauche de la fenêtre, fin est␣
 ↪le coin inférieur droit de la fenêtre et Y est le point objectif
    """
    x1,y1=deb
    x2,y2=fin
    resX = []
    resY = []
    chX = []
    chY = []
    x,y=x1,y1
    while x<x2:
        while y>y2:
            if EstPointDe(E,(x,y)):
                resX.append((x))
                resY.append((y))
                if estCheminConvexe(E,(x,y),Y,pas):
                    chY.append(y)
                    chX.append(x)
            y-=pas
        y=y1
        x+=pas
    plt.scatter(resY,resX)
    plt.scatter(chY,chX,color='r')
    plt.grid()
    plt.show()
```

Par exemple avec les chemins pour $Y = (0, 1.5)$ en rouge :



Cette fonction peut être utilisée afin de répartir les tâches entre les robots. Un robot plus proche d'un chemin direct sera plus rapide qu'un robot qui doit contourner plusieurs obstacles.

2.5.4 Implémentation du RRT

Nous avons récupéré la structure du code de Atsushi Sakai

[8]:
```
"""
Path planning Sample Code with Randomized Rapidly-Exploring Random Trees (RRT)
author: AtsushiSakai (@Atsushi_twi)
"""

import math
import random
import matplotlib.pyplot as plt
import numpy as np

show_animation = False

class RRT:
    """
    Class for RRT planning
    """
    class Node:
        """
        RRT Node
        """
        def __init__(self, x, y):
            self.x = x
            self.y = y
            self.path_x = []
            self.path_y = []
```

```
            self.parent = None

    def __init__(self, start, goal, obstacle_list, rand_area,
                 expand_dis=3.0, path_resolution=0.5, goal_sample_rate=5,␣
↪max_iter=500):
        """
        Setting Parameter
        start:Start Position [x,y]
        goal:Goal Position [x,y]
        obstacleList:obstacle Positions [[x,y,size],...]
        randArea:Random Sampling Area [min,max]
        """
        self.start = self.Node(start[0], start[1])
        self.end = self.Node(goal[0], goal[1])
        self.min_rand = rand_area[0]
        self.max_rand = rand_area[1]
        self.expand_dis = expand_dis
        self.path_resolution = path_resolution
        self.goal_sample_rate = goal_sample_rate
        self.max_iter = max_iter
        self.obstacle_list = obstacle_list
        self.node_list = []

    def planning(self, animation=True):
        """
        rrt path planning
        animation: flag for animation on or off
        """

        self.node_list = [self.start]
        for i in range(self.max_iter):
            rnd_node = self.get_random_node()
            nearest_ind = self.get_nearest_node_index(self.node_list, rnd_node)
            nearest_node = self.node_list[nearest_ind]

            new_node = self.steer(nearest_node, rnd_node, self.expand_dis)

            if self.check_collision(new_node, self.obstacle_list):
                self.node_list.append(new_node)

            if animation and i % 5 == 0:
                self.draw_graph(rnd_node)

            if self.calc_dist_to_goal(self.node_list[-1].x, self.node_list[-1].
↪y) <= self.expand_dis:
                final_node = self.steer(self.node_list[-1], self.end, self.
↪expand_dis)
```

```
                if self.check_collision(final_node, self.obstacle_list):
                    return self.generate_final_course(len(self.node_list) - 1)

            if animation and i % 5:
                self.draw_graph(rnd_node)

        return None  # cannot find path

    def steer(self, from_node, to_node, extend_length=float("inf")):

        new_node = self.Node(from_node.x, from_node.y)
        d, theta = self.calc_distance_and_angle(new_node, to_node)

        new_node.path_x = [new_node.x]
        new_node.path_y = [new_node.y]

        if extend_length > d:
            extend_length = d

        n_expand = math.floor(extend_length / self.path_resolution)

        for _ in range(n_expand):
            new_node.x += self.path_resolution * math.cos(theta)
            new_node.y += self.path_resolution * math.sin(theta)
            new_node.path_x.append(new_node.x)
            new_node.path_y.append(new_node.y)

        d, _ = self.calc_distance_and_angle(new_node, to_node)
        if d <= self.path_resolution:
            new_node.path_x.append(to_node.x)
            new_node.path_y.append(to_node.y)

        new_node.parent = from_node

        return new_node

    def generate_final_course(self, goal_ind):
        path = [[self.end.x, self.end.y]]
        node = self.node_list[goal_ind]
        while node.parent is not None:
            path.append([node.x, node.y])
            node = node.parent
        path.append([node.x, node.y])

        return path

    def calc_dist_to_goal(self, x, y):
```

```
        dx = x - self.end.x
        dy = y - self.end.y
        return math.hypot(dx, dy)

    def get_random_node(self):
        if random.randint(0, 100) > self.goal_sample_rate:
            rnd = self.Node(random.uniform(self.min_rand, self.max_rand),
                            random.uniform(self.min_rand, self.max_rand))
        else:  # goal point sampling
            rnd = self.Node(self.end.x, self.end.y)
        return rnd

    def draw_graph(self, rnd=None):
        plt.clf()
        # for stopping simulation with the esc key.
        plt.gcf().canvas.mpl_connect('key_release_event',
                                     lambda event: [exit(0) if event.key ==␣
↪'escape' else None])
        if rnd is not None:
            plt.plot(rnd.x, rnd.y, "^k")
        for node in self.node_list:
            if node.parent:
                plt.plot(node.path_x, node.path_y, "-g")

        for (ox, oy, size) in self.obstacle_list:
            self.plot_circle(ox, oy, size)

        plt.plot(self.start.x, self.start.y, "xr")
        plt.plot(self.end.x, self.end.y, "xr")
        plt.axis("equal")
        plt.axis([-2, 15, -2, 15])
        plt.grid(True)
        plt.pause(0.01)

    @staticmethod
    def plot_circle(x, y, size, color="-b"):  # pragma: no cover
        deg = list(range(0, 360, 5))
        deg.append(0)
        xl = [x + size * math.cos(np.deg2rad(d)) for d in deg]
        yl = [y + size * math.sin(np.deg2rad(d)) for d in deg]
        plt.plot(xl, yl, color)

    @staticmethod
    def get_nearest_node_index(node_list, rnd_node):
        dlist = [(node.x - rnd_node.x) ** 2 + (node.y - rnd_node.y)
                 ** 2 for node in node_list]
        minind = dlist.index(min(dlist))
```

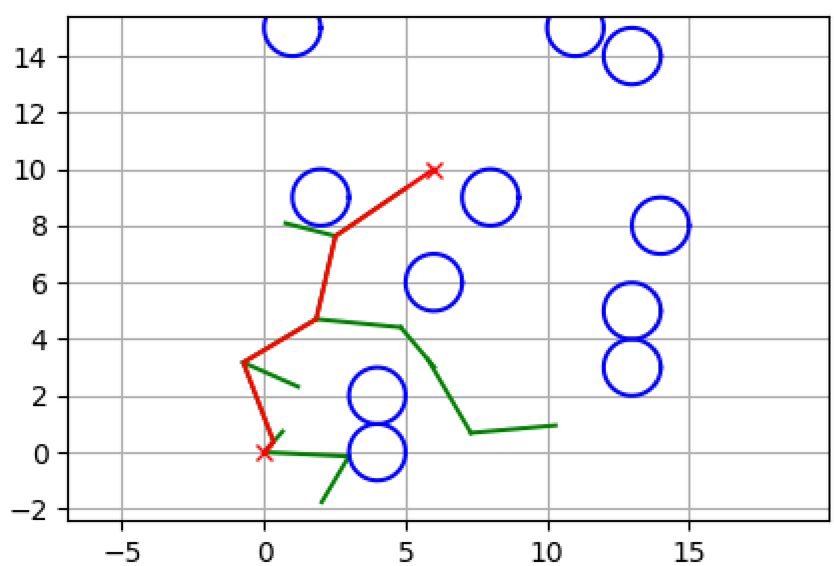
```
        return minind

    @staticmethod
    def check_collision(node, obstacleList):

        if node is None:
            return False

        for (ox, oy, size) in obstacleList:
            dx_list = [ox - x for x in node.path_x]
            dy_list = [oy - y for y in node.path_y]
            d_list = [dx * dx + dy * dy for (dx, dy) in zip(dx_list, dy_list)]

            if min(d_list) <= size ** 2:
                return False  # collision

        return True  # safe

    @staticmethod
    def calc_distance_and_angle(from_node, to_node):
        dx = to_node.x - from_node.x
        dy = to_node.y - from_node.y
        d = math.hypot(dx, dy)
        theta = math.atan2(dy, dx)
        return d, theta
```

Nous avons rajoutés les fonctions ci dessous :

[9]:
```
def Position_Robot(n,rad):
    """
    TODO : Fonction qui à remplacer par celle de vision qui positionne les␣
 ↪robots sur le terrain

    Genere la position des robots aléatoirement
    """
    res = []
    for i in range(0,n):
        res.append((random.randint(0,15),random.randint(0,15),rad))
    return res

def main(rx=0, ry=0, gx=6.0, gy=10.0,obstacleList=[]):

    # ====Search Path with RRT====
    # Set Initial parameters
    rrt = RRT(start=[rx, ry],
              goal=[gx, gy],
```

```
                rand_area=[-2, 15],
                obstacle_list=obstacleList)
    path = rrt.planning(animation=show_animation)
    if path is None:
        # Chemin introuvable
        return (False,[])
    else:
        print(path)
        # Chemin trouvé
        rrt.draw_graph()
        plt.plot([x for (x, y) in path], [y for (x, y) in path], '-r')
        plt.grid(True)
        plt.show()
        return (True,path)
```

Avec : main(obstacleList=Position_Robot(11,1)), on obtient :



3 Bibliographie

Artificial Intelligence
Chapitre 9 : PathFinder
Lien
-- Christian König
Description d'un algorithme RRT

Extended Team Description for RoboCup 2019
Partie 2 : Path Planning
Lien

-- Nicolai Ommer, Andre Ryll, Mark Geiger
Introduction d'un autre algorithme et critique du RRT

Extended Team Description Paper
Partie 2 : Path Planning
Lien
-- Andreas Wendler, Tobias Heineken
Critique du RRT et calcul de trajectoires en plus des chemins

Immortals 2020 Extended Team Description Paper
Partie 3.3 : Analyzing
Lien
-- Omid Najafi, MohammadAli Ghasemieh, Mehran Khanloghi, AmirMahdi Matin, AliReza Mohammadi, and AmirMahdi Torabian
Visualisation et analyse des chemins trouvés pas RRT

MRL Extended Team Description 2020
Partie 2.1 : Simple motion Planning
Lien
-- Meisam Kasaeian Naeini, Amin Ganjali Poudeh, Alireza Rashvand, Arghavan Dalvand, Ali Rabbani Doost, Moein Amirian Keivanani, SeyedEmad Razavi, Saeid Esmaeelpourfard, Erfan Fathi, Sina Mosayebi, and Aras Adhami-Mirhosseini
Introduction d'un algotithme plus simple basé sur des paraboles

Parabola Through Four Points
Lien
Démonstration de l'algorithme basé sur les paraboles