Team 358

Problem B

The Best Strategy for Penalty Kicks

Abstract

Exploring the optimal penalty kicks strategies has practical significance for football players. In this paper, we study the motion of a spinning soccer ball based on aerodynamics during its flight. Taking air buoyancy, gravity, air resistance and Magnus force into consideration, we provide the motion equations of the ball and solve the set of ordinary differential equations numerically. Given any initial velocity value and initial angular velocity value and direction, we can obtain the trajectory of soccer. We can also calculate the range of the reasonable direction angles θ and ϕ in the spherical coordinate system which can reach upper corners. Taking $v_0 = 25m/s, \vec{\omega} = 15\vec{k}$ as an example, we find the the range of θ is $[76.7°, 79.8°]$ and the range of ϕ is $[75.5°, 84.5°]$ and $[100.0°, 110.5°]$. Fixing $\vec{\omega}$, we gain the minimum value of initial velocity $v_{0min} = 20.17m/s$. We then consider two modes of defense of the goalkeeper:vertical movement defense and diving save. We firstly present the success probability distribution of defense by Monte Carlo Simulation. Subsequently we analysis the cases that the shooting player will definitely avoid the goalkeeper by finding the dead angles.It's found that the dead angle area is two symmetrical rectangles. Taking $v_0 = 25m/s, \vec{\omega} = 15\vec{k}$ as an example again, the range of θ is $[76.95°, 78.95°]$and the range of ϕ is $[75.5°, 75.9°]$ and $[110.3°, 110.6°]$.
Keywords: upper corner, Aerodynamics, Ordinary differential equation

Contents

1 Introduction

1.1 Background

Penalty kick, awarded when an offence punishable by a direct free kick is committed by a player in their own penalty area or a draw after extra time occurs , is sometimes seen in soccer matches. When a penalty kick is taken, a soccer player will take a shot at the goal from the penalty mark defended only by the opposing team's goalkeeper. As penalty kick sometimes plays an critical role in scores, soccer players might emphasize their training in penalty kick skills. The skills involve not only kick skills but also some strategies to mislead the opposing goalkeeper.

The force and flight trajectory is always analysed based on aerodynamics. Learning the rule of soccer's movement have practical significance for kickers to improve their success rate of penalty kicks. And some researches on soccer movement have be conducted by previous scholars.

1.2 Problem Restatement

When a penalty kick is taken, one player takes a single shot which is defended by one goalkeeper of the opposing team. The shooting player kicks the ball at the penalty mark, which is centered 11 meters from the goal. The goal is regarded as a rectangle 7.32 meters wide and 2.44 meters tall. The shooting player prefers to aim for an upper corner of the goal to avoid being defended. Our main task is to consider the initial ball velocity and spin for a successful shot to an upper corner. We also have to consider how to create the optimal initial ball velocity and spin to have the best probability to avoid the goalkeeper.

1.3 Problem Analysis

Firstly we need to set up the situation for kicking a penalty. The image of the penalty area is shown below:

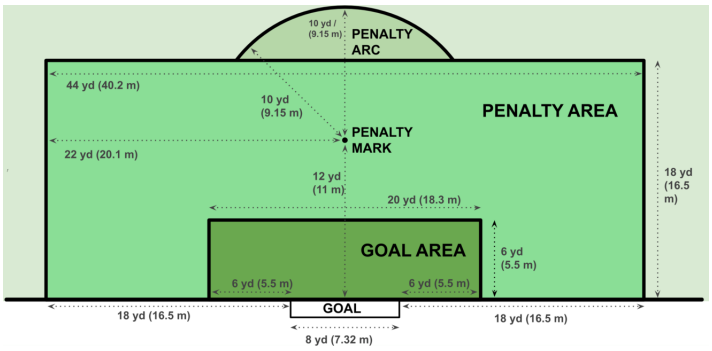

Figure 1: The penalty area[1]

To solve the first question, we have to define the upper corners of the goal. In general, we divide the area of the goal into nine equal parts, and consider the upper left corner and the upper right corner as the upper corners, which are shown in the figure below.

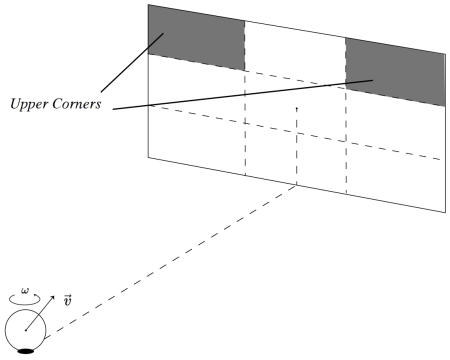

Figure 2: The upper corners

In order to obtain the initial ball velocity and spin for a successful shot to an upper corner, we have to analyze the motion of the ball and find the trajectory by analyzing the forces based on aerodynamics and solving dynamic equation system. The reasonable initial velocity and spin angular velocity should be a range.

To solve the second question, we have to consider the defense of the goalkeeper. Firstly we consider the probability of the goalkeeper's success in defending different areas. The shooting player tends to kick the ball towards the areas with a relatively lower probability of being defended successfully. Then we consider the areas that cannot be defended by the goalkeeper which will ensure the success of shooting.

2 Assumptions and Notations

2.1 Assumptions

- The parameters of soccer are determined according to standards set by the FIFA(International Federation of Association Football).
- Assuming that both the kicker and the goalkeeper are high level football players, which means that the kicker can accurately hit the ball to the target position and there is no possibility that the goalkeeper predict a totally wrong direction of the ball.
- The goalkeeper stands in the middle of the goal line at the start.
- We assume that the air is even and the situation is at normal temperature and pressure as air density varies at different temperatures and pressures.
- Environment does not change much during the flight of the ball, so we do not consider the change of temperature and the influence of natural wind.

2.2 Notations

Table 1: **Notations**

Symbol	Definition	Numeric Value
r	The radius of soccer	108mm
m	The mass of soccer	430g
g	The gravitational acceleration	9.8m/s^2
ρ_0	The mass density of air	1.27kg/m^3
C_a	An air resistance coefficient	0.4
H	The height of the goal	2.44m
W	The width of the goal	7.32m

3 Model

3.1 The Flight Trajectory of a Spinning Soccer

In order to find out the proper initial ball velocities and spins which will result in a successful shot to an upper corner,we firstly consider the flight trajectory of a spinning soccer. In addition to being affected by gravity, the ball also interacts with the air during the flight. In our Model, the interaction of the football with the air includes air resistance, air buoyancy and the Magnus force generated by rotation.

Before solving the problem, we establish a spatial Cartesian coordinate system. The penalty mark is the coordinate origin, while the positive direction of y-axis points to the center of the goal. When facing the goal, the positive direction of x-axis points to the right, and that of z-axis is straight up vertically. The example image is shown below:

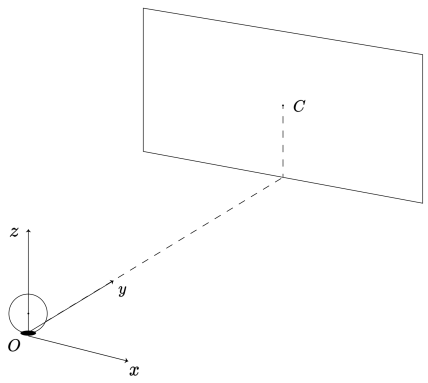

Figure 3: The spatial Cartesian coordinate system

3.1.1 Air Buoyancy and Gravity

As both air buoyancy and gravity are constants in the vertical direction, it is easy to gain their expressions. According to Archimedes principle, air buoyancy can be calculated:

$$\vec{F}_{float} = -\rho_0 \vec{g} V = -\frac{4}{3}\pi r^3 \rho_0 \vec{g} \tag{1}$$

$$\vec{G} = m\vec{g} \tag{2}$$

3.1.2 Air Resistance

The relationship between air resistance and flying speed can be obtained through a simplified model. We consider a round disk which is the maximum cross-section of the sphere and is orthogonal to the direction of velocity[2]. The example image is as follows:

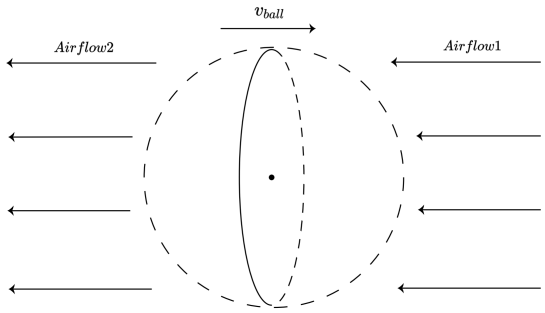

Figure 4: The airflow on both sides of the disk

The airflow on the right is not passed through by the ball, whose relative flow velocity is $v_1 = v$ and gas pressure is p_1 , while that on the left is blocked by the ball with the relative flow velocity $v_2 = 0$ and gas pressure p_2. From Bernoulli's principle, we can find an equation:

$$p_1 + \frac{1}{2}\rho_0 v_1^2 = p_2 + \frac{1}{2}\rho_0 v_2^2 \tag{3}$$

The pressure difference between the two sides is:

$$p_1 - p_2 = \frac{1}{2}\rho_0 v^2 \tag{4}$$

The effective area of action of pressure is $S = \pi r^2$, so the resistance to the disk is:

$$F_f = (p_1 - p_2)S = \frac{1}{2}\rho_0 S v^2 \tag{5}$$

This is a simplified expression of resistance. In reality, the situation is more complicated because the ball is not a disk, and the air rubs against the surface of the ball. We introduce an air resistance coefficient C_a to correct the equation, and according to relevant document[3], we find that $C_a = 0.4$. Finally, we gain the vector form of the equation:

$$\vec{F}_f = -\frac{1}{2}C_a \rho_0 S v^2 \frac{\vec{v}}{v} = -0.2\pi r^2 \rho_0 v \vec{v} \tag{6}$$

3.1.3 Magnus Force

Due to the combined influence of the spinning of the soccer ball and air viscosity, a circulation is generated around the soccer ball. On the side where the air flow is in the same direction as the direction of spinning, the air flow rate accelerates, while on the other side in the opposite direction, the air flow rate slows down.

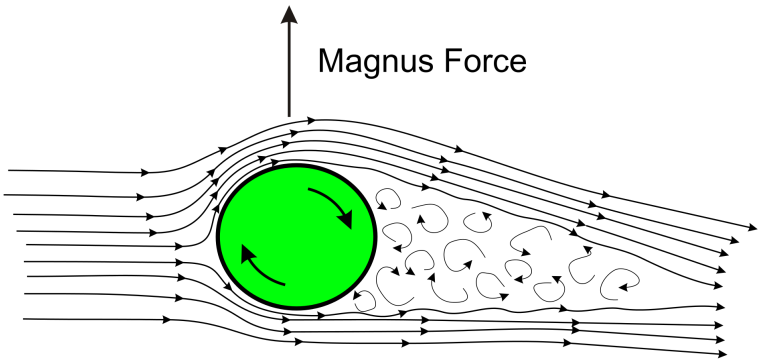

Figure 5: The Magnus effect[4]

According to Bernoulli principles, the pressure decreases on the side where the flow rate increases, while the pressure rises on the side where the flow rate slows down[5]. As a result, the pressure difference between the two sides exerts a lateral force on the football called the Magnus force. From Kutta–Joukowski theorem, we obtain the expression of Magnus force[6]:

$$\vec{F}_m = -\frac{8}{3}\pi r^3 \rho_0 \vec{\omega} \times \vec{v} \tag{7}$$

3.1.4 The Effect of Air Resistance on Ball Spin

Due to the tangential friction between the ball and the air as it rotates, the angular velocity of the soccer actually changes with time rather than being a constant. Like the normal friction which impacts the velocity of the ball, the effect of tangential friction can be written as $\frac{dw}{dt} = -kw$, where k is a coefficient related to radius of ball, density of air and Reynolds number of air. So the law of angular velocity changing with time is $\omega = \omega_0 e^{-kt}$.

As the rate of angular velocity change is proportional to the moment of tangential friction and the moment is cross product of radius and friction, the coefficient k is proportional to the cube of the radius. The radius of soccer is $0.108m$, so the order of magnitude of k is the negative cube of 10. Plus, the flight time of the football is less than one second. So the final angular velocity is less than ω_0 and larger than $\omega e^{-10^{-3}} = 0.999\omega_0$, which can be regarded as a constant ω_0. Therefore, we can conclude that the change of angular velocity due to tangential friction can be neglected.

3.1.5 Derivation of Motion Equation

As we have obtained the expressions of gravity, air buoyancy, air resistance and Magnus force, the motion equation can be present:

$$\begin{aligned} m\tfrac{d^2\vec{r}}{dt^2} &= \vec{G} + \vec{F}_{float} + \vec{F}_f + \vec{F}_m \\ &= m\vec{g} - \tfrac{4}{3}\pi r^3 \rho_0 \vec{g} - 0.2\pi r^2 \rho_0 v \vec{v} - \tfrac{8}{3}\pi r^3 \rho_0 \vec{\omega} \times \vec{v} \end{aligned} \tag{8}$$

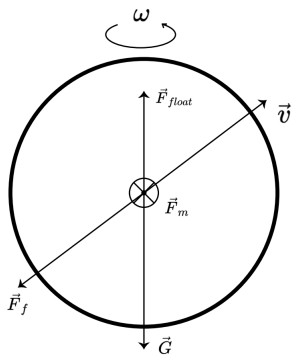

Figure 6: Forces analysis

We decompose velocity and angular velocity into three directions ($\vec{i}$,$\vec{j}$,$\vec{k}$ are the unit vectors of x,y,z axes):

$$\begin{aligned} \vec{v} &= v_x\vec{i} + v_y\vec{j} + v_z\vec{k} \\ \vec{\omega} &= \omega_x\vec{i} + \omega_y\vec{j} + \omega_z\vec{k} \end{aligned} \tag{9}$$

Then we can decompose the vector motion equation into three component forms:

$$\begin{cases} m\frac{d^2x}{dt^2} = -0.2\pi r^2\rho_0\sqrt{{v_x}^2+{v_y}^2+{v_z}^2}\cdot v_x + \frac{8}{3}\pi r^3\rho_0\left(v_y\cdot\omega_z - v_z\cdot\omega_y\right) \\ m\frac{d^2y}{dt^2} = -0.2\pi r^2\rho_0\sqrt{{v_x}^2+{v_y}^2+{v_z}^2}\cdot v_y + \frac{8}{3}\pi r^3\rho_0\left(v_z\cdot\omega_x - v_x\cdot\omega_z\right) \\ m\frac{d^2z}{dt^2} = -0.2\pi r^2\rho_0\sqrt{{v_x}^2+{v_y}^2+{v_z}^2}\cdot v_z + \frac{8}{3}\pi r^3\rho_0\left(v_x\cdot\omega_y - v_y\cdot\omega_x\right) - mg + \frac{4}{3}\pi r^3\rho_0 g \end{cases} \tag{10}$$

3.1.6 Solution to the Motion Equation

The motion equation is a set of second-order ordinary differential equations and there is no symbolic solution. We write Python codes to solve it numerically, and we can easily obtain the flight trajectory as long as giving the initial velocity $\vec{v}_0$ and angular velocity $\vec{\omega}_0$.

According to Ravenscroft's research[7], the ball initial velocity can reach 23.9 $\pm$1.2(**Unit** : m/s)(average player). So our calculation is based on this data. Here we provide an example to show the trajectory ($\vec{v}_0 = -3.40\vec{i} + 19.27\vec{j} + 4.16\vec{k}$(**Unit** : m/s), $\vec{\omega}_0 = 15\vec{k}$(**Unit** : rad/s)):

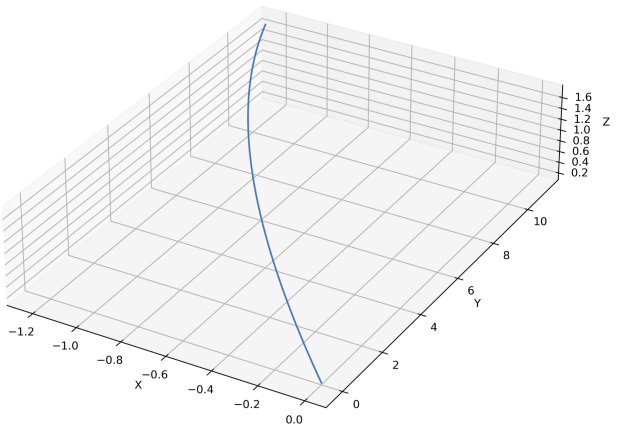

Figure 7: A trajectory as an example

Due to the value and direction of initial velocity, as well as the value and direction of initial angular velocity are unknown, it is difficult to judge the landing point of the trajectory. To simplify the solution, we suppose the value of initial velocity is constant and the direction is variable, and suppose the angular velocity is also a constant. In this case, we can get the range of initial velocity direction angles that the ball can be kicked into the upper corner.

Considering that the value of initial velocity is $v_0 = 25(\textbf{Unit} : m/s)$, we transform Cartesian coordinates into spherical coordinates and using θ and ϕ to express the direction:

$$\begin{cases} v_{x0} = u_0 \sin\theta \cos\varphi \\ v_{y0} = u_0 \sin\theta \sin\varphi \\ v_{z0} = u_0 \cos\theta \end{cases} \tag{11}$$

When it comes to the angular velocity of spinning, we present two possible cases specifically.

Case1: The angular velocity contains only z-direction component We fix the angular velocity at $\vec{\omega} = 15\vec{k}(\textbf{Unit} : rad/s)$. The goal area is divided into nine rectangles of the same shape, with the upper corners including the upper left corner and the upper right corner. According to the above initial conditions, we change θ and ϕ to obtain the range of direction angles in which the ball can be shot into the upper corners.

We solve the problem by writing Python programs, and finally we obtain an image of the reasonable direction angles.

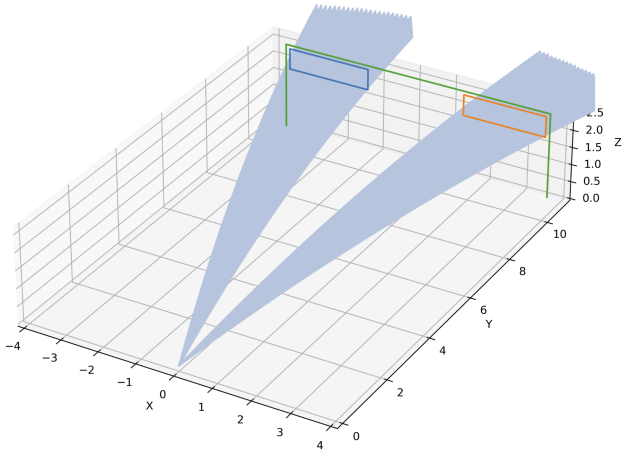

Figure 8: The trajectories with reasonable initial direction angles with $\vec{\omega} = 15\vec{k}$

The range of θ is $[76.7°, 79.8°]$ and the range of ϕ is $[75.5°, 84.5°]$ and $[100°, 110.5°]$.

We then change the value of the initial velocity, and traversed a set of initial velocity values through Python programming to obtain the minimum initial velocity that can reach the upper corners. We finally find out that when the value of initial velocity is less than $v_{0min} = 20.17m/s$, there is no possibility of the ball to hit the upper corners.

Case2: The angular velocity contains x,y,z-direction components We assume the angular velocity as $\vec{\omega} = 6\vec{i} + 6\vec{j} + 6\vec{k}(\textbf{Unit} : rad/s)$. we obtain an image of the reasonable direction angles as follow.

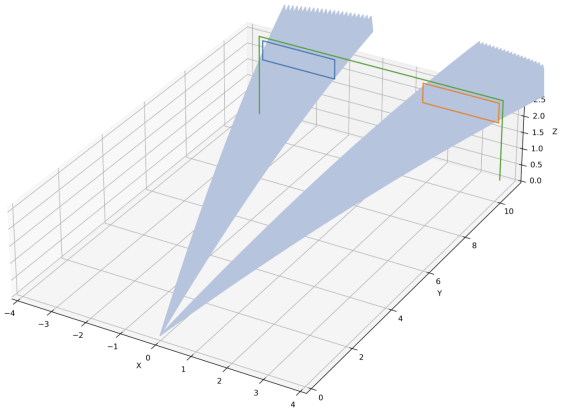

Figure 9: The trajectories with reasonable initial direction angles with $\vec{\omega} = 6\vec{i} + 6\vec{j} + 6\vec{k}$

The range of θ is $[75.29°, 79°]$ and the range of ϕ is $[73.5°, 84.0°]$ and $[98°, 108.5.5°]$. When the value of initial velocity is less than $v_{0min} = 20.18m/s$, there is no possibility of the ball to hit the upper corners.

It is worth noting that, comparing with $\vec{\omega} = 15\vec{k}$, the trajectory of $\vec{\omega} = 6\vec{i} + 6\vec{j} + 6\vec{k}$ is more asymmetric, which is the effect of the initial angular velocity.

To sum up, in the condition of the given value of initial velocity and initial angular velocity, the ball can reach the upper corners if the shooting player kicks it in the range of orientation angles shown in the image above. As the reasonable orientation angles can't be expressed by mathematical language such as equation and inequalitywe only give the range of reasonable orientation angles here. The specific value range of the direction angle can be obtained by using the code in the appendix.

3.2 The Best Way of Avoiding the Goalkeeper and Making the Goal

To simplify the calculation, the height of the goalkeeper is assumed to be 1.90m, which is recognized as the best height of goalkeepers, and denoted as h. According to relevant document [8],the distance between shoulders and chin is the $\frac{1}{2}$ length of head and the length of head is $\frac{1}{8}$ of the whole body. Therefore the height of the goalkeeper's shoulder center, denoted as h_l, is:$h_l = h \cdot \frac{8-1-1/2}{8} = 154.375$cm. Also from the document[8], the length of a person's arms outstretched is equal to his height.So the length of goalkeeper's arm, denoted as l,is $l = h \cdot \frac{1}{2} = 0.95$m.

After getting all the required parameterswe can discuss several different defense methods of goalkeeper, and calculate the corresponding range that the goalkeeper can reach.

In the following discussion, we will analysing two defense modes:vertical movement defense and diving save. Vertical movement defence means the goalkeeper stands still, jumps up or squats down with arms opened. Diving save means the goalkeeper jump left or right to block the ball.

3.2.1 Probability Distribution of Successful Defense

In order to avoid the goalkeeper and make the goal, the first thing is to figure out the probability distribution of the goalkeeper's defense. Kickers should shoot the area with relatively small defense probability. In the following discussion, we will analyse the probability distribution of Successful Defense and visualize it.

From the former model, the minimum reasonable velocity is $20.17m/s$, and the we can calculate the corresponding maximum flight time $0.65s$ with Python program. So we assume the maximum

time for goalkeeper to block the soccer is $t_{max} = 0.65s$.

Vertical movement defense Let's first discuss the vertical movement defense. If the goalkeeper stand still with arms opened, then in the middle upper part of the goal, the reachable area consists of a circle drawn by arms.If the goalkeeper squats down, then the area under the former circle can all be reached. With a maximum descent acceleration g, the time consumption of squatting down is $\sqrt{2\frac{h_l}{g}} = 0.56$s, which is feasible because it is shorter than t_{max}.

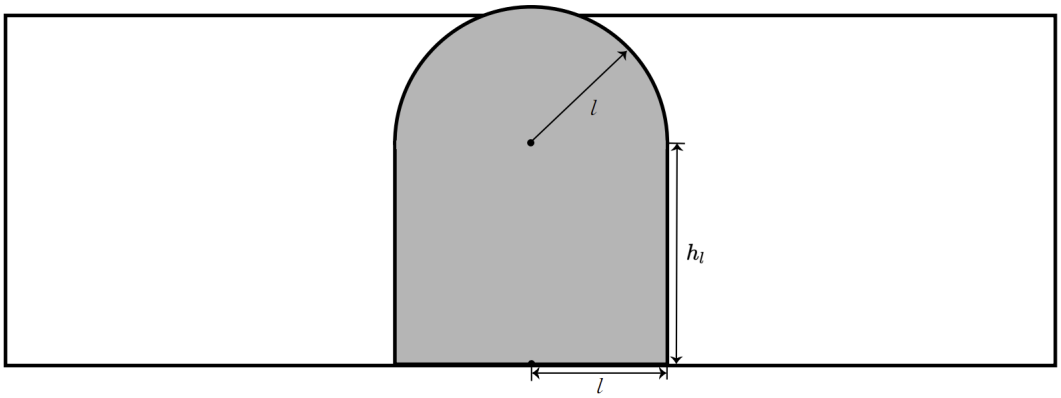

Figure 10: The area reached by vertical movement defense without jump

From the above graph and results, we can find that with a jump higher than 0.90m, the goalkeeper can reach all the area above the bar. Then we analyze the possibility of a jump higher than 0.9m and in reasonable time. A professional athlete can jump 1 meter high. And we can estimate the time as $\sqrt{2 \cdot g \cdot \Delta h} = 0.43$s, which is a reasonable time. Therefore, a jump higher than 0.90m is feasible. So in vertical movement defense mode, the reachable area inside the goal is a rectangle $2l = 1.9$m wide and $H = 2.44$m tall.

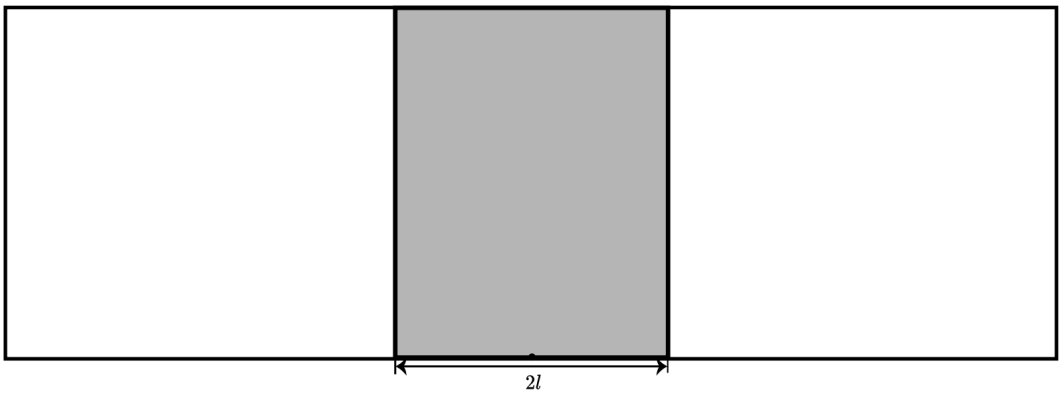

Figure 11: The area reached by vertical movement defense

Diving save Next, we discuss the diving save. Again we assume the goalkeeper can jump to 1 meter high. And the acceleration distance, which is replaced with the estimated squat height 40cm,is

denoted as s_a. Then the take off speed v_0 can be calculated as $v_0 = \sqrt{2g\Delta h} = 4.43\text{m/s}$. The force F_N on the ground when taking off and the acceleration time t_0 satisfy the equation set:

$$\begin{array}{l} v_0^2 = 2(\frac{F_N}{m} - g)S_a \\ v = (\frac{F_N}{m} - g)t_0 \end{array} \tag{12}$$

After solving the equation set, we get:$\frac{F_N}{m} = 34\text{m/s}^2, t_0 = 0.181\text{s}$. The goalkeeper may jump either to the left or to the right. The angle between the acceleration force from the ground F_N and the ground is denoted as α. Consider F_N and the gravity mg, we get the horizontal and vertical acceleration as $a_x = \frac{F_N}{m} \cdot cos\alpha, a_z = \frac{F_N}{m} \cdot sin\alpha - g$.

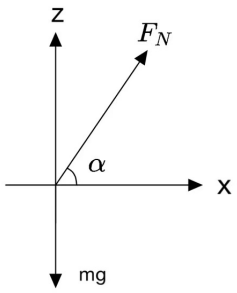

Figure 12: Force analysis of goalkeeper when taking off

After multiplying t_0, we get the taking off speed:

$$\begin{cases} \mathrm{v_x} = \frac{F_N}{m} \cdot cos\alpha \cdot \mathrm{t} \\ \mathrm{v_y} = (\frac{F_N}{m} \cdot sin\alpha - g) \cdot \mathrm{t} \end{cases} \tag{13}$$

So the coordinates of the goalkeeper is:

$$\begin{cases} x = v_x t \\ y = y_0 + v_y t - \frac{1}{2} g t^2 \end{cases} \tag{14}$$

where y_0 is the initial height of the shoulders. Eliminating time t, we get the trajectory of goalkeeper's shoulder as:

$$y = y_0 + v_y \frac{x}{v_x} - \frac{1}{2} g \left(\frac{x}{v_x}\right)^2 \tag{15}$$

Up to now, all the variables such as v_x, x is determined only by the angle α. What we need to do now is find the α maximizes the area below the trajectory and also in reasonable time, which is relative to the reachable area by diving save mode.

The area can be calculated as:

$$S = \int_0^{x_t} \left(y_0 + v_y \frac{x}{v_x} - \frac{1}{2} g \left(\frac{x}{v_x}\right)^2\right) dx = y_0 x_t + \frac{1}{2} \frac{v_y x_t^2}{v_x} - \frac{1}{6} g \frac{x_t^3}{v_x^2} \tag{16}$$

where x_t is the largest x the goalkeeper's shoulder can reach in t_{max} and $x_t = v_x \cdot t_{max}$.

We writing python codes to get the α that maximizes S. $\alpha = 0.64rad$. And we can plot the trajectory as follow.

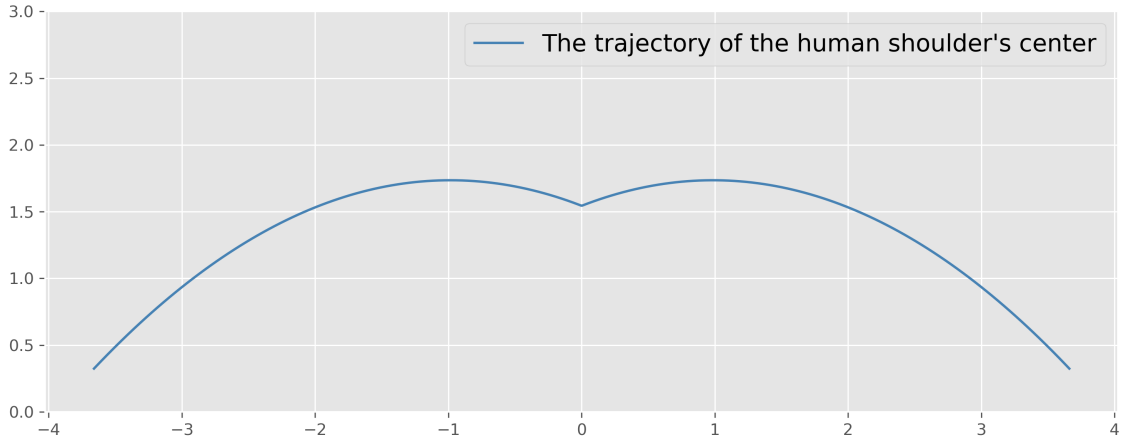

Figure 13: The trajectory of goalkeeper's shoulders

We can also obtain $x_t = 3.235$m.

Considering the arms of goalkeeper, we draw the farthest points arms can reach, and interpolate them. After interpolation, we can get the reachable area by diving save mode.

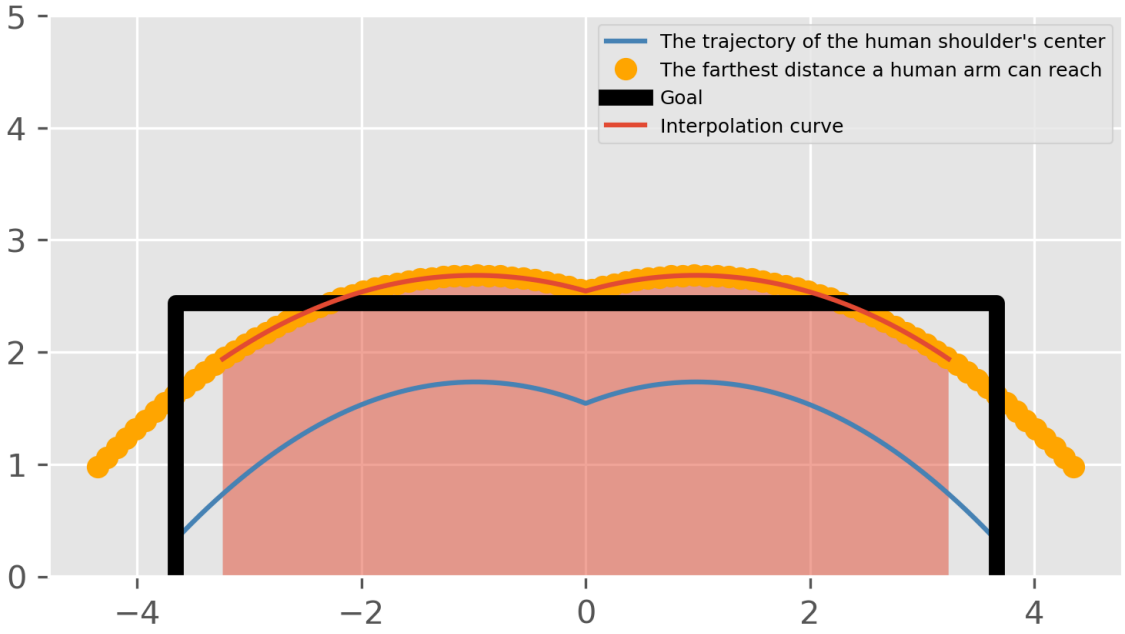

Figure 14: The area reached by diving save

Considering the two defense modes, the probability distribution is the sum of them. We assume that the success probability of a goalkeeper's defense is a two-dimensional normal distribution in x and z directions.

At the same time, we give different weights to different areas in the goal, considering that different defensive approaches of goalkeepers have different defense areas and they overlap. The weighting method is based on the overlaps of the different defensive areas. The more times the defense areas

overlap, the greater the weight is. So, we can define $r(x, z)$ to denotes the success possibility:

$$r(x, z) = nf_{xz}(x, z) \tag{17}$$

Here, n denotes the number of times the defensive area overlaps. $f_{xz}(x, z)$ is the normal function on the xOz plane. We can use the Monte Carlo simulation method to study the the distribution of the success possibility, shown in Figure 15. The lighter the color appears, the higher the probability of successful defense is .

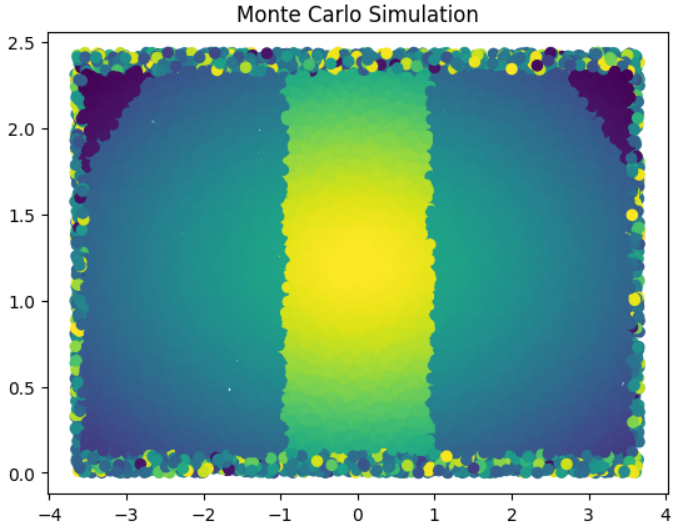

Figure 15: Distribution of success possibility

3.2.2 The Dead Angle of the Goalkeeper's Defense

Instead of seeking to shot the points with small probabilities that a goalkeeper can reach, we shot the points that goalkeeper can't reach any way due to constraints such as physical limitation. However, we neglect the time limit $t_{max} = 0.65$s as we are pursuing the absolute died angle area.

Also we discuss these two defense methods of goalkeeper, and calculate the corresponding range that the goalkeeper can reach. Then, we find the intersection of all the ranges that can be reached, and find the difference between the reachable range and the goal range, which is the dead angle zone.

Vertical movement defense The vertical movement defense mode is the same as above, so we directly use the result of the above model, that is , in vertical movement defense mode, the reachable area inside the goal is a rectangle $2l = 1.9m$ wide and $H = 2.44m$ tall.

Diving save However, the diving save mode is different from the above one. The integral upper limit x_t changes because there is no time limit.

We can calculate x_t with the following method.When $y = 0$, $x_t = \frac{v_x v_y + v_x\sqrt{v_y^2+2gy_0}}{g}$, so we get :

$$S = \int_0^{x_t} \left(y_0 + v_y \frac{x}{v_x} - \frac{1}{2} g \left(\frac{x}{v_x} \right)^2 \right) dx = y_0 x_t + \frac{1}{2} \frac{v_y x_t^2}{v_x} - \frac{1}{6} g \frac{x_t^3}{v_x^2} \tag{18}$$

We also write python codes to get the α that maximizes S. $\alpha = 0.92$rad. And we can plot the trajectory as follow.

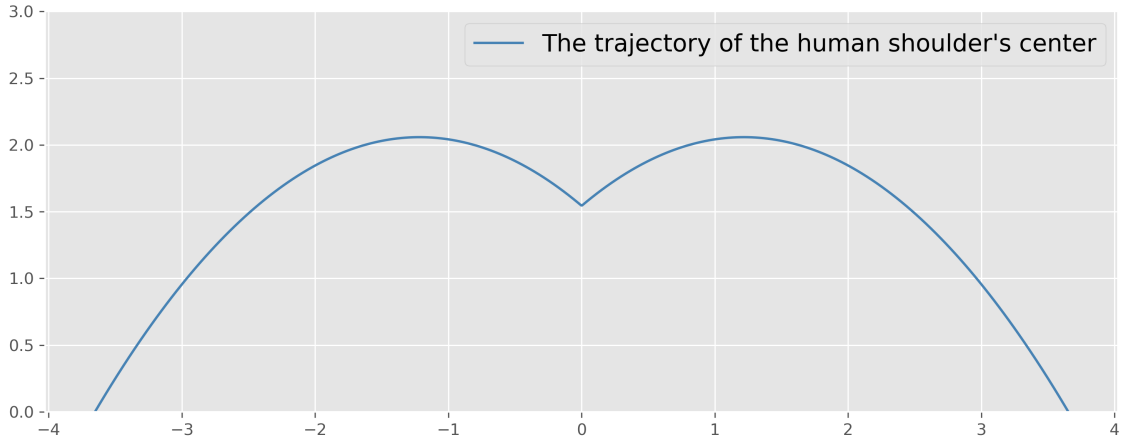

Figure 16: The trajectory of goalkeeper's shoulders in dead angle situation

We can get time of movement $t_s = \frac{x_t}{v_x} = 0.972$s.

Also, we draw the farthest points arms can reach, and interpolate them. After interpolation, we can get the reachable area by diving save mode.

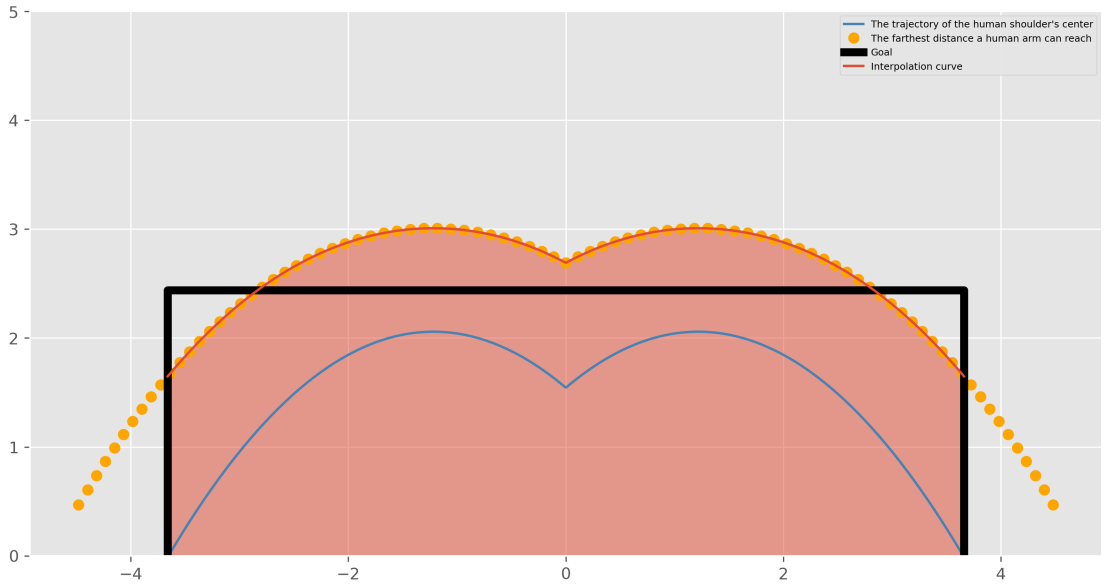

Figure 17: The area reached by diving save in dead angle situation

Considering the two defense methods, there are only two corners left inside the goal, which is the absolute dead angle area.

Let's have a look at the dead angle area:

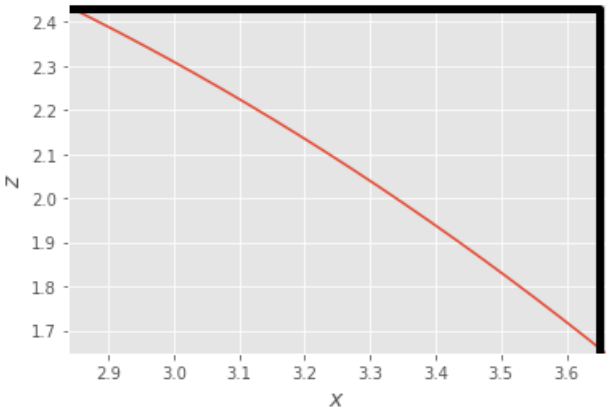

Figure 18: The dead angle area

It can be simplified as a triangle. The intersections of reachable area(or dead angle area) and the goal can be calculated with the interpolated area function. Their coordinates are $(\pm 2.84, 2.44)$,$(\pm 3.66, 1.648)$. And considering the radius of football, the actual dead angle area is a rectangle smaller than the triangle above, but it is still large enough for the ball to pass through.

Again we assume the angular velocity as $\vec{\omega} = 15\vec{k}(\textbf{Unit} : rad/s)$ and initial velocity as $v_0 = 25((Unit) : m/s)$. Using Python programs, we obtain an image of the reasonable direction angles.

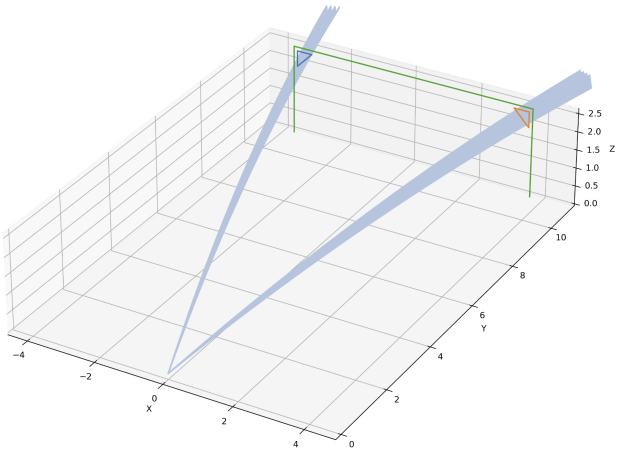

Figure 19: The trajectory of soccer with reasonable initial direction angles

The ball can reach the upper corners if the shooting player kicks it in the range of orientation angles shown in the image above. The specific value range of the direction angle can be obtained by using the code in the appendix. The range of reasonable orientation angles are as follows. The range of θ is $[76.95°, 78.95°]$ and the range of ϕ is $[75.5°, 75.9°]$ and $[110.3°, 110.6°]$.

4 Discussion

4.1 Advantages

- **We consider the issue comprehensively.**
  When analysing the force on soccer, we take forces into account as much detail as possible. And also, we analyse two defense modes of the goalkeeper.

- **The universality of our model is strong.**
  With a random initial conditions(including direction and value of velocity and angular velocity), we can calculate and obtain the trajectory of the soccer, and therefore determine whether the ball can enter the target area.

- **The solution is of high accuracy.**
  With suitable numerical analysis methods and Python programming, we obtain results of high accuracy. For example, when calculating the reasonable direction range, we traverse enough points to obtain the set of solution.

- **Our assumptions and parameters are close to actual fact.**
  All of our parameters are not our subjective assumptions, but are obtained based on official or recognized information.

- **Our visualization is illustrative.**
  During the analysis, we draw a lot of schematic diagrams to help understand and result diagrams to make the results more intuitionistic.

- **Our model is innovative.**
  We use the Monte Carlo simulation method to study the the distribution of the success possibility.

4.2 Disadvantages

- **Less convincing data**
  Although we endeavor to find more data about soccer players for making a better theoretical analysis, our data is still limited and cannot accurately measure the players, which may lead to the inaccuracy of the initial velocity interval.

- **Ignorance of some factors**
  To make our model easier to compute, we ignore some factors. For example, we neglect the variation of air density and simplify defensive strategy of the goalkeeper. Also, perhaps in the actual environment where the goal is located, the player does not have a good grasp of the speed and direction of ball, and therefore can't kick soccer to the target point.

5 Conclusion

After the analysis of the first question, we establish a model of the trajectory of the ball based on aerodynamics. With random initial velocity and angular velocity, we can obtain the trajectory of soccer. We find that the reasonable value of initial velocity should be more than $v_{0min} = 20.17m/s$. Given an angular velocity at $\vec{\omega} = 15\vec{k}$(**Unit:**rad/s), the range of θ is $[76.7°, 79.8°]$ and the range of ϕ

is $[75.5°, 84.5°]$ and $[100.0°, 110.5°]$. While given an an angular velocity at $\vec{\omega} = 6\vec{i}+6\vec{j}+6\vec{k}$, the range of θ is $[75.29°, 79°]$ and the range of ϕ is $[73.5°, 84.0°]$ and $[98°, 108.5.5°]$. We can also obtain the range of direction angles when given any set of initial velocity values and angular velocities through our codes.

When taking the defense of the goalkeeper into consideration, we consider two defense modes: vertical movement defense and diving save. We gain the success probability distribution of defense through Monte Carlo Simulation. We then consider the cases that the shooting player will definitely make the goal by finding out the areas that the goalkeeper cannot reach, which is found as a rectangle. Again we provide the angular velocity as $\vec{\omega} = 15\vec{k}(\textbf{Unit} : rad/s)$ and initial velocity as $v_0 = 25((Unit) : m/s)$. In this case the range of θ is $[76.95°, 78.95°]$and the range of ϕ is $[75.5°, 75.9°]$ and $[110.3°, 110.6°]$.

In all, if the shooting player follows the strategy provided by our task, he will have the best chance of avoiding the goalkeeper and making the goal.

References

[1] Wikimedia Commons. File:goal area and penalty area green.png — wikimedia commons, the free media repository, 2020. [Online; accessed 6-November-2022].

[2] F Li. Motion analysis and equation derivation of banana kicks. *SILLICON VALLEY*, 2013.

[3] M. J. Carré, S. R. Goodwill, and S. J. Haake. Understanding the effect of seams on the aerodynamics of an association football. *P I MECH ENG C-J MEC*, 219(7):657–666, 2005.

[4] Wikimedia Commons. File:sketch of magnus effect with streamlines and turbulent wake.svg — wikimedia commons, the free media repository, 2021. [Online; accessed 6-November-2022].

[5] Thorsten Kray, Jörg Franke, and Wolfram Frank. Magnus effect on a rotating soccer ball at high reynolds numbers. *Journal of Wind Engineering and Industrial Aerodynamics*, 124:46–53, 2014.

[6] D W Liu. Discussion of spherical flight trajectory anomalies. *COLLEGE PHYSICS*, pages 43–45, 1987.

[7] James C Ravenscroft and Nicholas P Linthorne. Effect of the location of the foot impact point on ball velocity in a soccer penalty kick. 2015.

[8] Joseph Sheppard. Anatomy: A complete guide for artists. 1992.

Appendix

A Codes

Code1

```
# calculating the range of theta and phi and the trajectory of the ball
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from scipy.integrate import odeint
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from math import radians

m = 0.4
k = 0.2*np.pi*1.29*0.11**2
g = 9.8
wx = 0
wy = 0
wz = 15
r = 0.108
G = 8/3*np.pi*1.29*0.11**3
v0 = 25
rho0 = 1.29


def diff_equation(sol_list, t):

    x, xx, y, yy, z, zz = sol_list
    return np.array([xx,
                     -k*np.sqrt(xx ** 2+yy ** 2+zz ** 2) *
                     xx+G*(yy*wz-zz*wy),
                     yy,
                     -k*np.sqrt(xx ** 2+yy ** 2+zz ** 2) *
                     yy+G*(zz*wx-xx*wz),
                     zz,
                     -k*np.sqrt(xx ** 2+yy ** 2+zz **2) *
                     zz+G*(xx*wy-yy*wx)-m*g+3/4*np.pi*r**3*rho0*g])


ax = plt.axes(projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

vx = []
vy = []
vz = []



def theta_phi_sweep(v0=25):
    p = []
    pmin=[]
    pmax=[]
    i = 0
    for theta in np.arange(76.6, 79.9, 0.1):
```

```
52          p.append(0)
53          pp = []
54          for phi in np.arange(70, 85,0.5):
55              vx0 = v0*np.sin(radians(theta))*np.cos(radians(phi))
56              vy0 = v0*np.sin(radians(theta))*np.sin(radians(phi))
57              vz0 = v0*np.cos(radians(theta))
58              t = np.linspace(0, 0.6, num=100)
59              sol0 = [0, vx0, 0, vy0, r, vz0]
60              result = odeint(diff_equation, sol0, t)
61              # plt.plot(t, result[:, 0], label='x')  # x
62              # plt.plot(t, result[:, 2], label='y')  # y
63
64              x = result[:, 0]
65              y = result[:, 2]
66              z = result[:, 4]
67              if np.argwhere(y > 11+r).size >0:
68
69                  id = np.argwhere(y > 11+r)[0]
70                  if z[id] > 1.7346666666666668 and z[id] < 2.332:
71                      if  -3.552<x[id] < -1.328 or 1.328 < x[id] < 3.552:
72                          ax.plot3D(x, y, z,color='lightsteelblue')
73                          pp.append(phi)
74                          vx.append(vx0)
75                          vy.append(vy0)
76                          vz.append(vz0)
77          for phi in np.arange(91,115,0.5):
78              vx0 = v0*np.sin(radians(theta))*np.cos(radians(phi))
79              vy0 = v0*np.sin(radians(theta))*np.sin(radians(phi))
80              vz0 = v0*np.cos(radians(theta))
81              t = np.linspace(0, 0.6, num=100)
82              sol0 = [0, vx0, 0, vy0, r, vz0]
83              result = odeint(diff_equation, sol0, t)
84
85              x = result[:, 0]
86              y = result[:, 2]
87              z = result[:, 4]
88              if np.argwhere(y > 11+r).size >0:
89
90                  id = np.argwhere(y > 11+r)[0]
91                  if z[id] > 1.7346666666666668 and z[id] < 2.332:
92                      if  -3.552<x[id] < -1.328 or 1.328 < x[id] < 3.552:
93                          ax.plot3D(x, y, z,color='lightsteelblue')
94                          pp.append(phi)
95                          vx.append(vx0)
96                          vy.append(vy0)
97                          vz.append(vz0)
98
```

```
            # if pp == []:
            #     print(theta)
            p[i] = pp
            if pp!=[]:
                pmin.append(pp[0])
                pmax.append(pp[-1])
            i+=1
        return [p,pmin,pmax]

print('$\\theta$ minimum:',np.arange(76.6, 79.9, 0.1)[1])
print('$\\theta$ maximum:',np.arange(76.6, 79.9, 0.1)[-2])
print('$\phi$ minimum:',min(theta_phi_sweep()[1]))
print('$\phi$ maximum:',max(theta_phi_sweep()[2]))
# draw rectangle
ax.plot([-7.32/2+r,-7.32/2+r,-7.32/6-r,-7.32/6-r,-7.32/2+r],
        [11,11,11,11,11],[2.44*2/3+r,2.44-r,2.44-r,2.44*2/3+r,2.44*2/3+r])
ax.plot([7.32/2-r,7.32/2-r,7.32/6+r,7.32/6+r,7.32/2-r],
        [11,11,11,11,11],[2.44*2/3+r,2.44-r,2.44-r,2.44*2/3+r,2.44*2/3+r])
ax.plot([-7.32/2,-7.32/2,7.32/2,7.32/2],
        [11,11,11,11],[0,2.44,2.44,0])
plt.xlim(-4,4)
plt.ylim(0,11)

plt.gca().set_box_aspect((8, 11, 2.5))
plt.show()
```

Code2

```
# calculating the minimun value of velocity
v0min=0
for v0 in np.arange(20.1,20.25,0.01):
    p= theta_phi_sweep(v0)[0]
    for i in p:
        if i!=[]:
            v0min=v0
            print('v0min:',v0)
            break
    if v0min!=0:
        break
```

Code3

```
# Distribition
import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np
import sympy as sp
from scipy.optimize import fmin
from scipy.interpolate import lagrange
import scipy.interpolate as interpolate
import matplotlib.patches as mpathes

sa=0.4
a=34.33
g=9.8
hl=1.54375
t=0.181
r=0.108
arm=0.95#arm length

# %%
def goal(x,z):

    if -7.32/2+r<x<7.32/2-r and r<z<2.44-r:
        return True
    else:
        return False

def vmd(x,z): #vertical movement defense
    l=1.9/2

    if -l<x<l and z<2.44:
        return True
    else:
        return False

def diving_save(x,z):

    alpha=0.67324219
    def fun(x):
        y0=hl
        vx=a*np.cos(alpha)*t
        vy=(a*np.sin(alpha)-g)*t
        y=y0+vy*x/vx-1/2*g*(x/vx)**2
        return y

    xx=np.linspace(-3.66,3.66,1000)

    xs,ys = sp.symbols('x y')
    dfun=sp.diff(fun(xs),xs)

    x_array = np.linspace(0,3.66,50)
```

```
    y_array=[]
    for member in x_array:
        y_array.append(fun(member))

    temp = []

    for i in range(50):
        fun_x = float(dfun.evalf(subs={xs:x_array[i],ys:y_array[i]}))
        temp.append(fun_x)

    dfun_array = np.array(temp)
    #tan(theta)


    rx=[]
    ry=[]
    for i in range(50):
        k=-1/dfun_array[i]
        if k>0:
            rx.append(x_array[i]+1*arm/np.sqrt(k**2+1))
            ry.append(y_array[i]+k*arm/np.sqrt(k**2+1))
        else:
            rx.append(x_array[i]-1*arm/np.sqrt(k**2+1))
            ry.append(y_array[i]-k*arm/np.sqrt(k**2+1))

    true_rx=[]
    minus_rx=[]
    true_ry=[]
    for i in range(50):
        if rx[i]>0 or rx[i]==0:
            true_rx.append(rx[i])
            true_ry.append(ry[i])
            minus_rx.append(-rx[i])


    ff=interpolate.splrep(true_rx,true_ry,s=0)

    if z<interpolate.splev(abs(x),ff,der=0):
        return True
    else:
        return False


# %%
```

```
import random
import numpy as np
from scipy.stats import norm
%matplotlib inline
import matplotlib.pyplot as plt



def score(x,z):
    global w
    if goal(x,z):
        w=1
        if vmd(x,z):
            w+=1

        if diving_save(x,z):
            w+=1
        w=w*norm.pdf(x,loc=0,scale=7.32/2)*norm.pdf(z,loc=1.22,scale=1.22)
    return w



# %%
color=[]
xx=[]
zz=[]


# %%
for i in range(20000):
    x=np.random.uniform(-7.32/2,7.32/2)
    z=np.random.uniform(0,2.44)
    color.append(score(x,z))
    xx.append(x)
    zz.append(z)

# %%


plt.scatter(xx, zz, c=color,s=35)
plt.title('Monte Carlo Simulation')

plt.show()
```