Final report of internship

JINQING XU

**Variable Definition Table**

|  |  |
| --- | --- |
| Variables | Definition |
|  | The normal vector of the tablet plane. |
| trial | Starting after clicking start button and ending when final lift up after a successful selection. |
| attempt | Starting after touch down and ending when lift up no matter selecting the target successfully or not. |
| entryFreq | The frequency of entry. |
| entry | The start point of the movement is outside the target and the end point of the movement is inside the target. |
|  |  |
|  |  |

**The Communication of leap motion and android**

* Challenge

There are still no SDK for the connection of leap motion and android.

So the two devices collect data separately. However, we need to find the relationship between 2D data and 3D data.

* Method

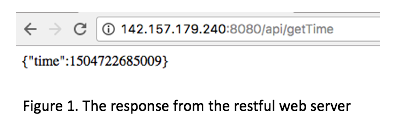
Since the android and leap motion programs run simultaneously, we can use the timestamp to locate the start and end of a trial in the leap motion data.

* Difficulties

We use the timestamp on the laptop to represent the time of a frame from leap motion data. And we use the system time of android to represent start-time and final-lift-up-time of a trial from android data. But the time of laptop and android cannot always be simultaneously. Even if I do adjustment, the difference of timestamp will be greater as time goes on due to the difference of clock frequency in different devices. We found that the clock frequency in tablet is somehow slower than that in laptop.

* Solution

The solution is to use the internet to find the difference of timestamp dynamically. Firstly, I started a restful web server on the laptop. Whenever a GET request is received, it will respond with the current timestamp of the laptop as shown in Figure 1.



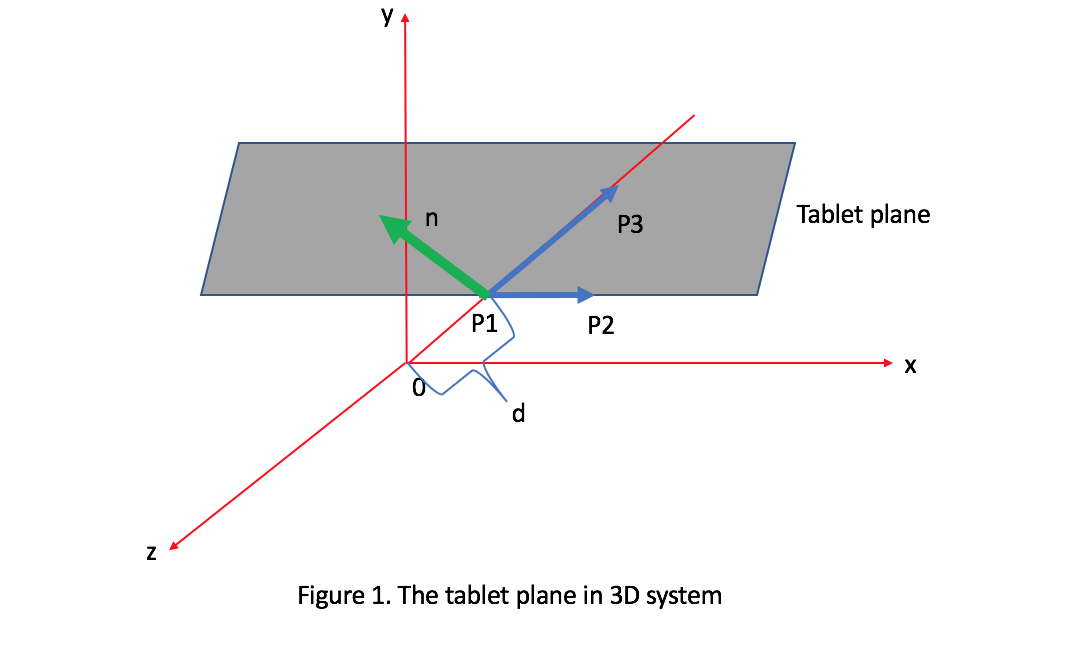
Secondly, at the initial stage of android program, it sends GET request to the web server to get the current timestamp of laptop. By comparing the timestamp from android system to that from laptop, we can find the timestamp difference of android relative to laptop. After that, each time we record timestamp in android system, we add that difference.

There is a tricky problem. After we send a request, there will be much network delay before we receive the response. To ensure the accuracy of timestamp from laptop, we need to take that delay into consideration. Before we send the request, we use beginClock to record the start time. After we receive response from the server, endClock is used to represent the end time. So Round-Trip Time is endClock minus beginClock. Since we get the system time of android after being responded, the delay should be half of RTT. Then the real timestamp from laptop should add up the delay on the base of the response value. Finally, we get the difference of timestamps from two devices.

(add one plot)

**Measures from Mackenzie:**

* Variables that need to be defined in advance
* The normal vector of the tablet plane.



1. suppose , and are three points on the tablet.

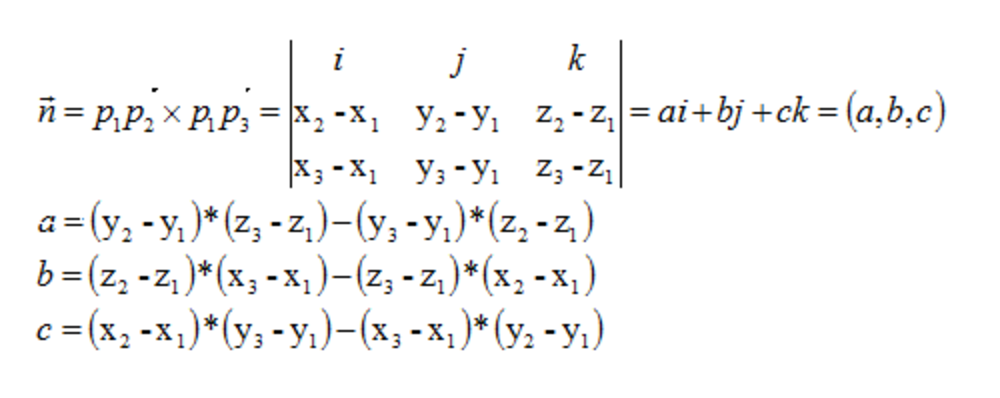
Let = (, , ) = ( , , ) = ( , , )

Then

Vector = (- , -, -)

vector = (-, -, - )

Let be the normal vector of tablet.



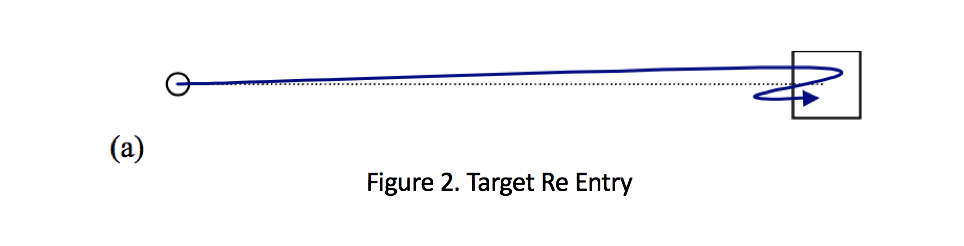
1. Let the distance between O and P1 be d

Then = ( 0,0,-d ) = ( 1,0,-d ) = ( 0,1,-d-1 )

1. Finally, we got the normal Vector of tablet = (0,1,1)

* Target Re-entry (See codes in Appendix A)
* Definition

TRE means first enter the target, then left, and re-enter again. TRE is detected when the finger moves on the tablet. An entry behavior means the start point of the movement is outside the target and the end point of the movement is inside the target. Let **entryFreq** be the frequency of entry. We accumulate **entryFreq** and each time when it is above zero, we minus its value by 1 to get the re-entry frequency since re-entry means entry again.



* Measures

1. TRE

The average re-entry frequency per attempt.

1. First TRE

The re-entry frequency of the first attempt.

* Movement Direction Change (See code in Appendix B)
* Definition

The direction change during a movement. There are changes in X, Y and Z direction.

The leap motion measures X, Y and Z value of the fingertip. When the current x is smaller than the previous one, the direction is **Left.** When the current y is smaller than the previous one, the direction is **Down.** When the current z is smaller than the previous one, the direction is **Backward**.

We record the previous direction and each time the current direction does not equal to the previous direction, a Movement Direction Change happens. Replace the previous direction with the current direction at the end of each loop.

* Task Plane Crossing (See code in Appendix C)
* Definition

The number of times that the paths of the finger cross the task plane.

If the finger is initially above the task plane and then below the plane, a Task Plane Crossing happens. We accumulate the frequency of Task Plane Crossing in a trial.

* How to judge a point is above or below the plane.

Firstly, we need to calculate the function for the task plane. To get the normal vector of the plane , we need to find two vectors vertical to it.

Since the normal vector for tablet plane is vertical to the normal vector of the task plane,

We got \* = 0 (1)

Since the direction vector of the task axis is vertical to the normal vector of the task plane,

We got \* = 0 (2)

Then we can get the normal vector (, , )

Let the 3D position of the start point be S(,,)

The function of the task plane is

\*(x-) + \*(y-)+\*(z-) = 0 (3)

To judge whether a point P ( , , ) is above or below the plane, we put its value into the function (3)

D=\*(-) + \*(-) + cn\*(-) = 0 (4)

If D > 0, P is above the task plane

If D==0, P is on the task plane

If D < 0, P is below the task plane

* Movement Offset ( see code in Appendix D)
* Movement Offset means the mean deviation of finger from the task plane. The deviation is measured by the distance from the finger point to the task plane. When the finger is above the task plane, the deviation is negative. And when the finger is below the task plane, the deviation is positive.
* Calculate the distance between a point P and a plane.

Let the line passing through P be l. Let the intersection point of l and plane be C. The distance between P and the plane equals to that between P (, , ) and C (.

1. get the function of l.

We have already calculated the normal vector (, , ) of the task plane when we calculate task axis crossing. It is also the direction vector of l.

The function of l:

=k (5)

1. get the intersection point C.

From (5), we can use k to represent x, y, z

x=\*k+ (6)

y=\*k+ (7)

z=\*k+ (8)

Put x, y, z into the function of task plane (3)

We get

K= (9)

Then put (9) into (6),(7),(8)

We can get C (.

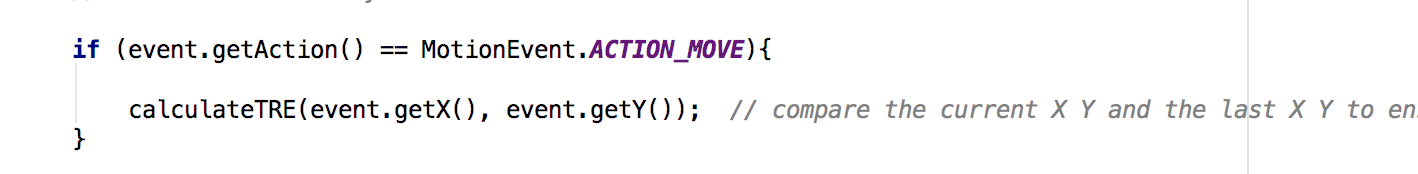
1. calculate the distance between P and C.

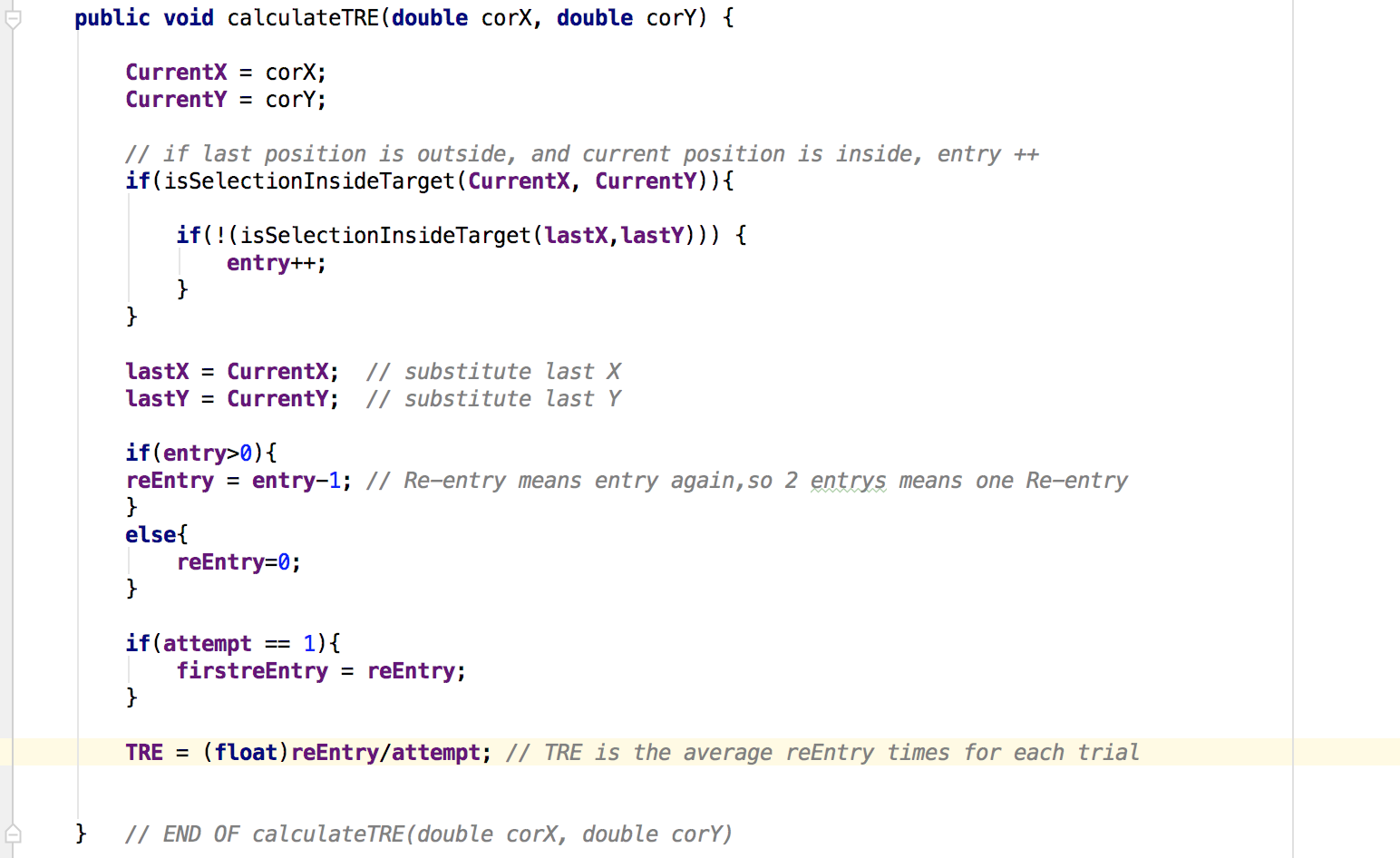
D=

* Movement Error ( see code in Appendix E)

Appendix A

CalculateTRE function is in the android code and it’s called when a move event is detected.

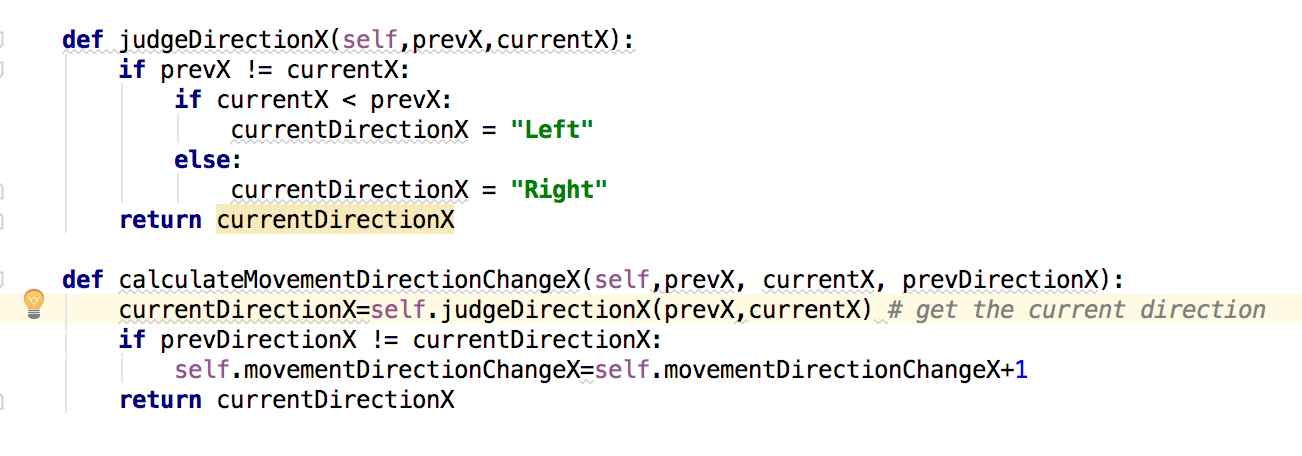
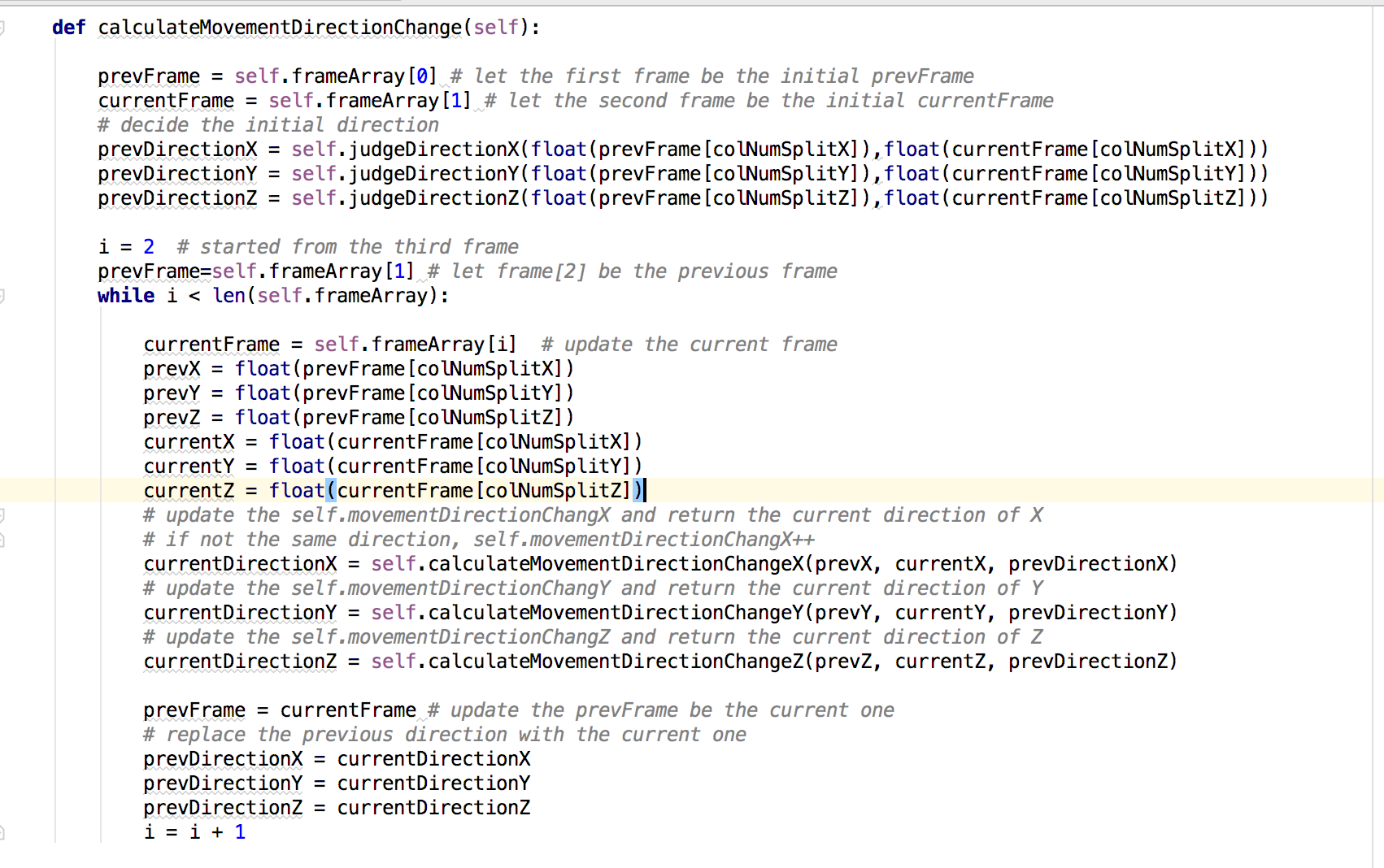




Appendix B

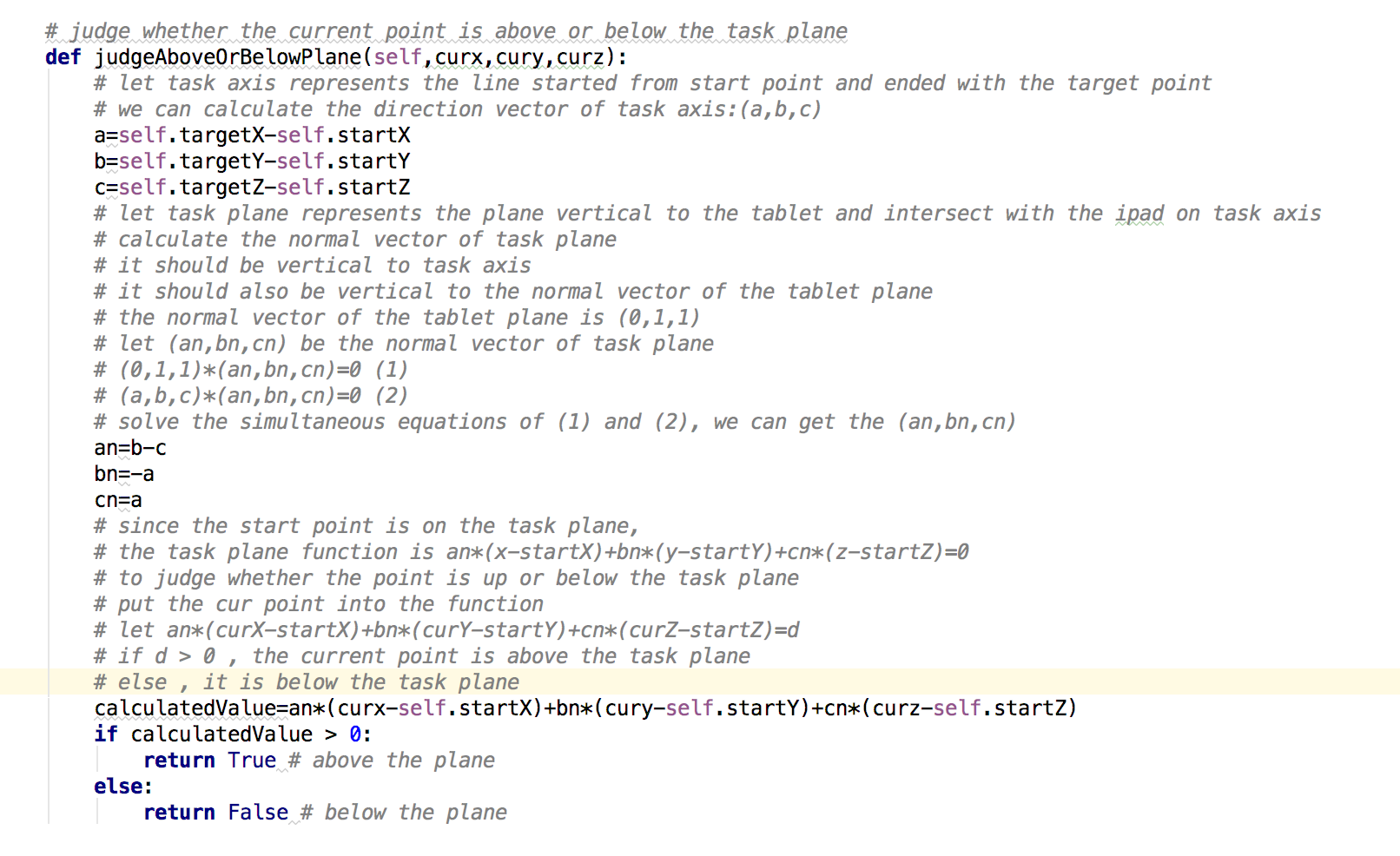
Firstly, using the first and second frame to judge the initial direction.

Then let frameArray[1] as the initial prevFrame, the loop begin with the frameArray[2].

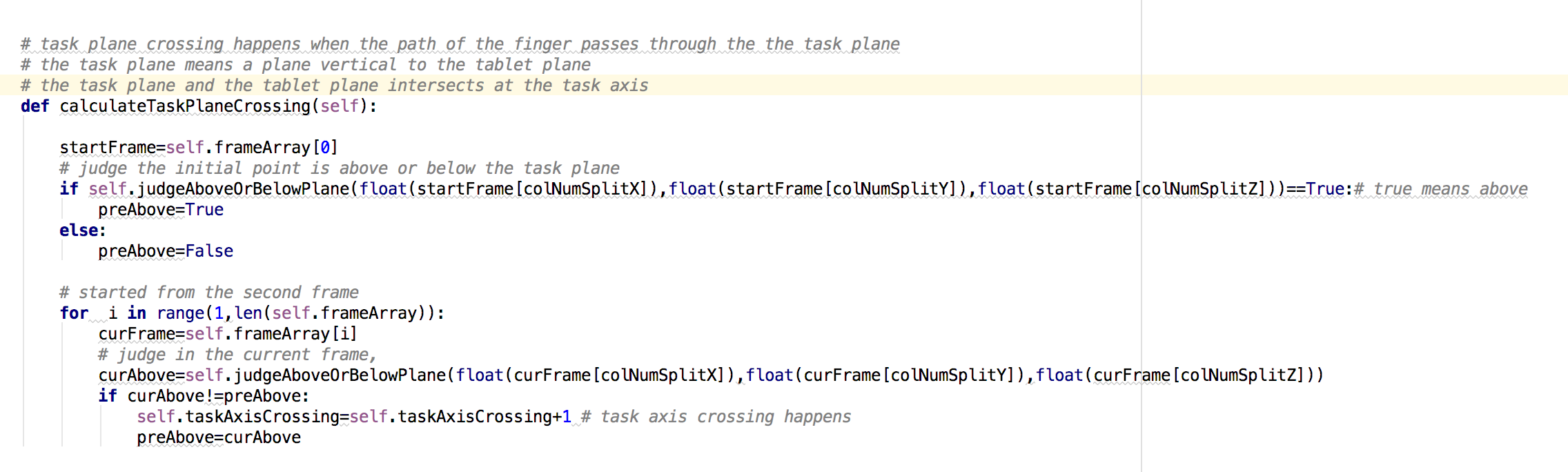


Appendix C

* Judge whether a point is above or below the plane

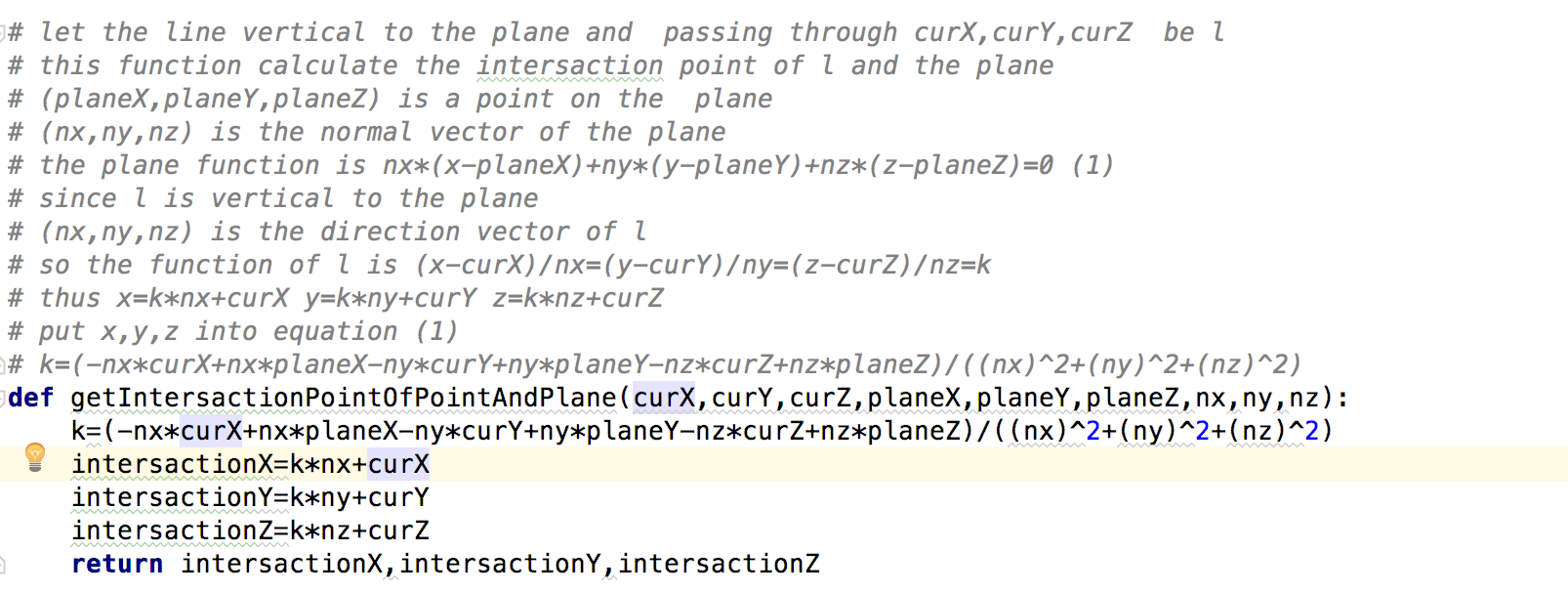


* Calculate Task Plane Crossing

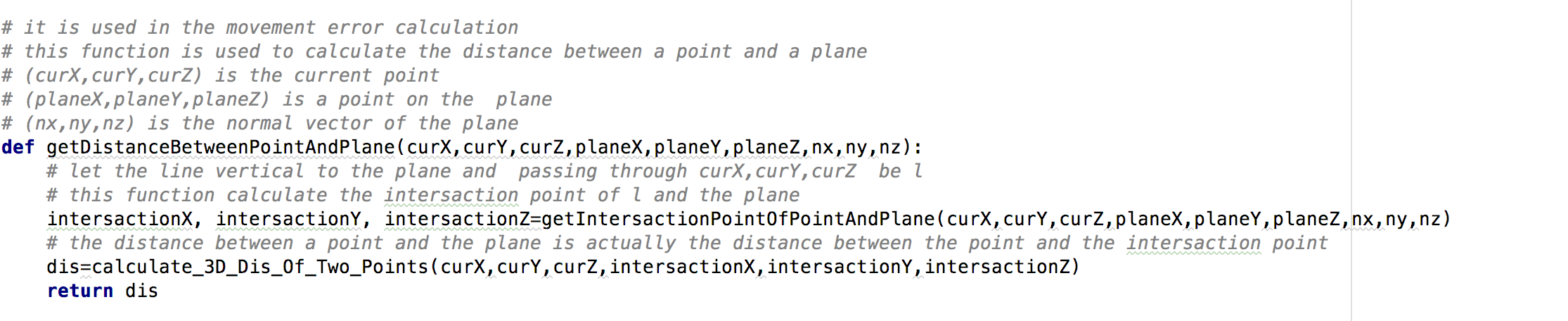


Appendix D

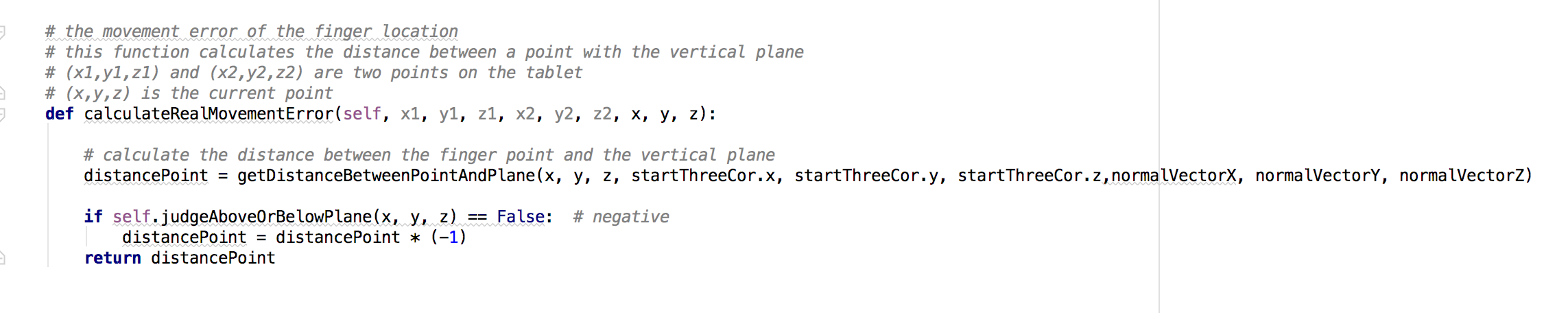
* get the intersection point C



* get the distance between C and P



* get the real distance (positive/negative) between the finger and the task plane



* get offset ( the average of mean distance)

