Our goal is to simulate approach of an object to grasshopper's eye.

Imagine this situation:
we have 6cm object (parameter size set to 0.06m) that is far far away from grasshopper and that is approaching toward the eye with speed of 2m/s for example.
At the beginning grasshopper doesn't see it at all. After some time it starts seeing it as a small black dot at horizont. (that can be one black pixel at screen). As it approaches, size of object on iPad screen starts to be greater and greater. When it gets at distance of 10cm (iPad distance) the size of object on iPad should be exactly the same as of real object. As it approaches further, size of the object on the iPad should be greater than actual size of object that we simulate (6cm). When it hits the eye basically it covers whole eye of grasshopper and it has theoretically infinite size on ipad screen.
Since iPad can not display infinite size of object we have some limitation in simulation.
Simulation is actually working like this: First we imagine object at great distance and we move imaginary object with constant velocity (usually 2m/s) toward the grasshopper. Each frame we calculate angle Theta for triangle: center of the object - grasshopper's eye - top of the object. Look at the Image 1.

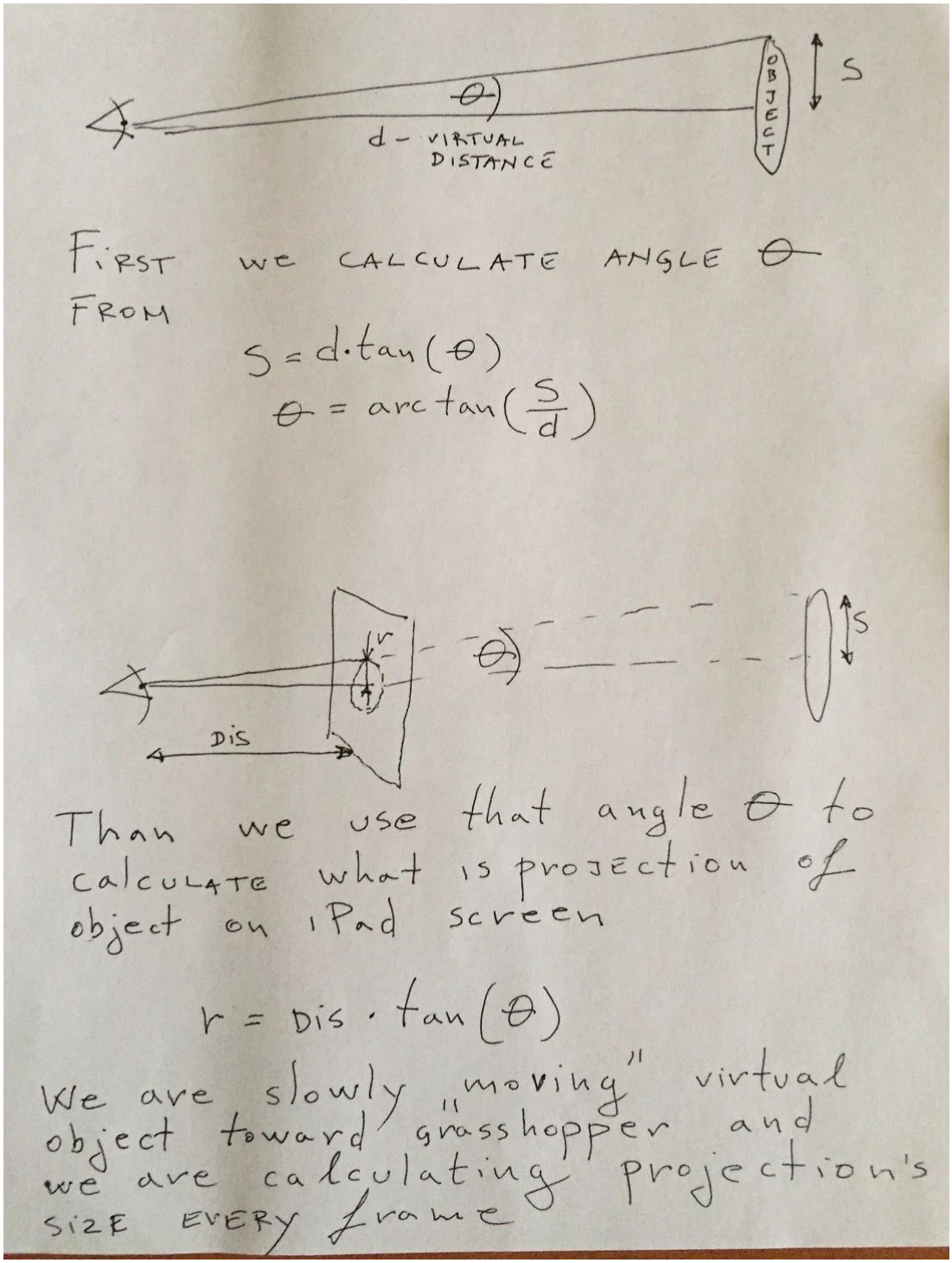


Image 1.

After we calculate that angle Theta, and if we know distance from grasshopper to the iPad, we can calculate size of projection of object on iPad screen ("r" is radius of object on iPad screen). See bottom of Image 1.
Than we draw that object on iPad screen and we continue to next frame.
First problem is that iPad is finite size (14.7cm x 19.7cm) and we can not simulate actual impact to grasshopper since simulated object will fill the screen of iPad even before it hits the eye of grasshopper (See Image 2)

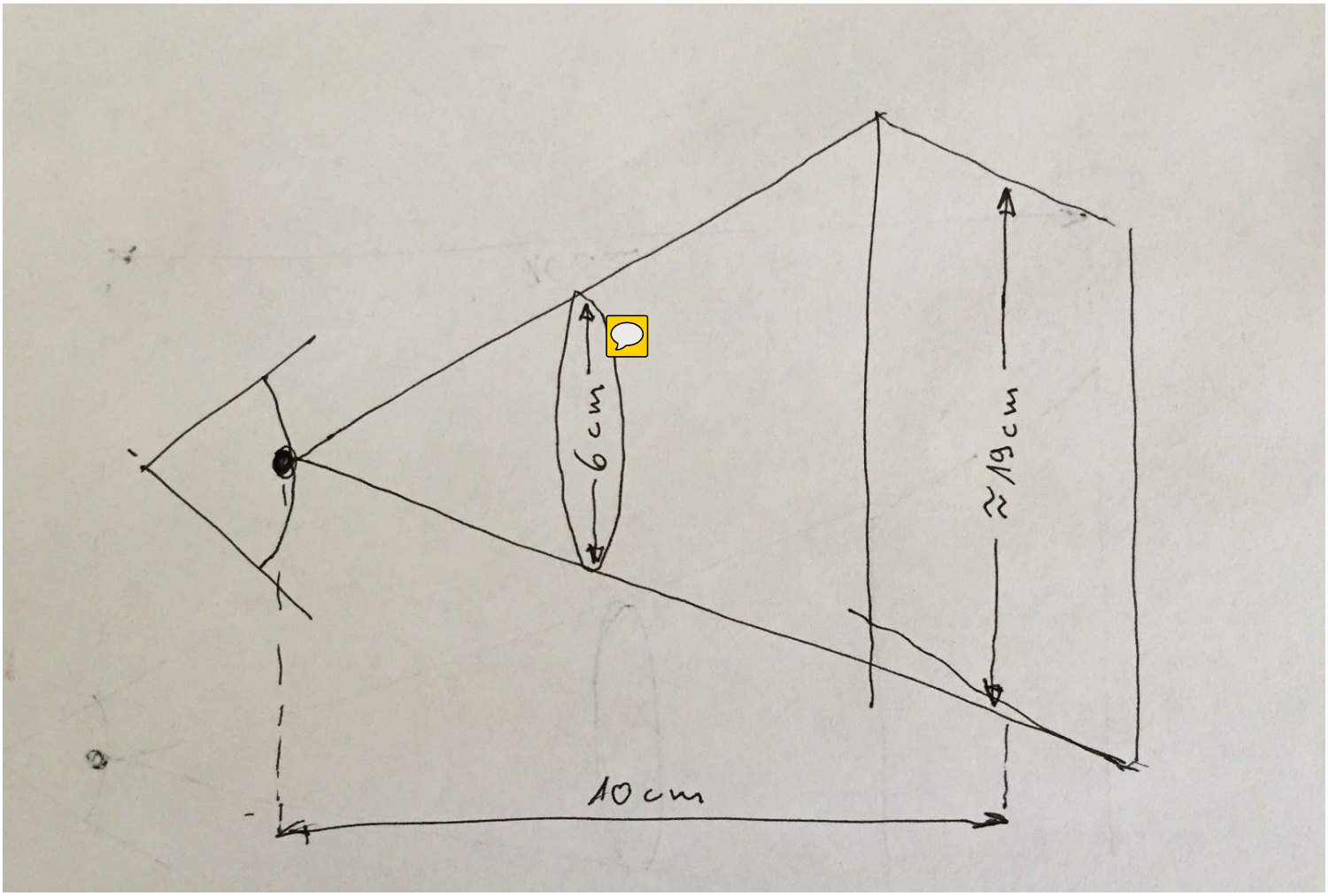


Image 2

Good thing is that this problem is present in any simulation with display/monitor no matter how big is the screen. So other papers have the same problem. (hope they are aware of it). Only way to simulate impact properly is to actually hit the grasshopper with some object.
The question that this raises is: Is this valid simulation? I think that it is valid simulation. Since grasshoppers are living in nature/grass and common situation is that their visual field is occluded by grass and other stuff in nature. I don't think that they will change behaviour if they are looking through grass "window" to an approaching object in contrast to behaviour when they don't have occluded visual field. (This is hypothesis, you can not state that without proving it. It can be nice paper if you can prove that with different sizes of screens or different distances to the screen.)

Second problem is synchronization of recording: Graphical simulation of looming is based on some virtual time that is driven by iPad's clock. When I start simulation (virtual time zero) I also start recording spikes from SpikerBox. Even though I did everything to start those two at the same time, those two things are different processes in iPad and we can not guarantee that they will start at the same time (initialization of recording file depends on IOS system and it can be delayed by unknown time). Since we treat those two processes as if they are synchronous we will introduce some error in our measurements (Δt_{rec}).
Beside that our recording hardware is not perfect so we will have some delay in SpikerBox (Δt_{SB}) and audio card of iPad (Δt_{iPad}). If you sum up all those errors you will get cumulative error (Δt_{err}) that is present in final analysis/graphs and that makes our life miserable. Take a look at Image 3 below that sum up those errors/delays.

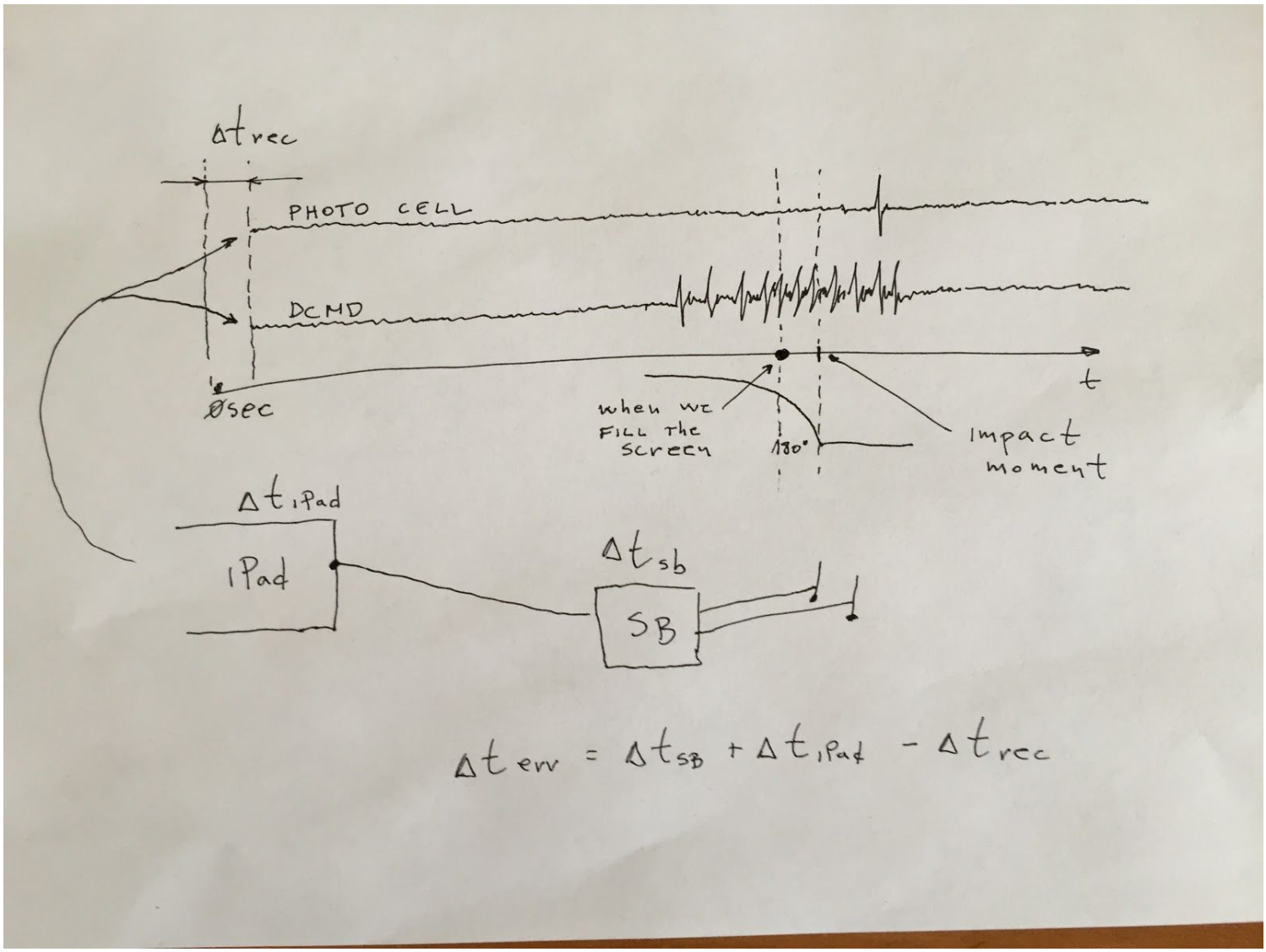


Image 3

On Image 3 you can see virtual time that drives graphic simulation does not start at the same time as recordings. I showed here as if recordings are delayed by time Δt_{rec}. Actually recordings can start even before simulation but our formula for cumulative error Δt_{err} is still valid. Also, you can see on timeline black dot that represent moment when our object fill up the screen of iPad and you can see mark for impact moment after that.

Below timeline I graphically represent angle Theta that goes to 180 degrees (even though we can not display object with Theta equals to 180 degree since that is circle with infinite radius. Take look at Image 1 and Image 2).

Beside that I symbolically showed our hardware and delay that it introduces (Δt_{iPad}, Δt_{SB}).

First error that we make in analysis is to treat recordings as if they start at the same time as graphical simulation. Take a look at Image 4

In image 4 you can see that we aligned recordings with virtual time in our analysis and in that way we just ignored delay Δt_{rec}.

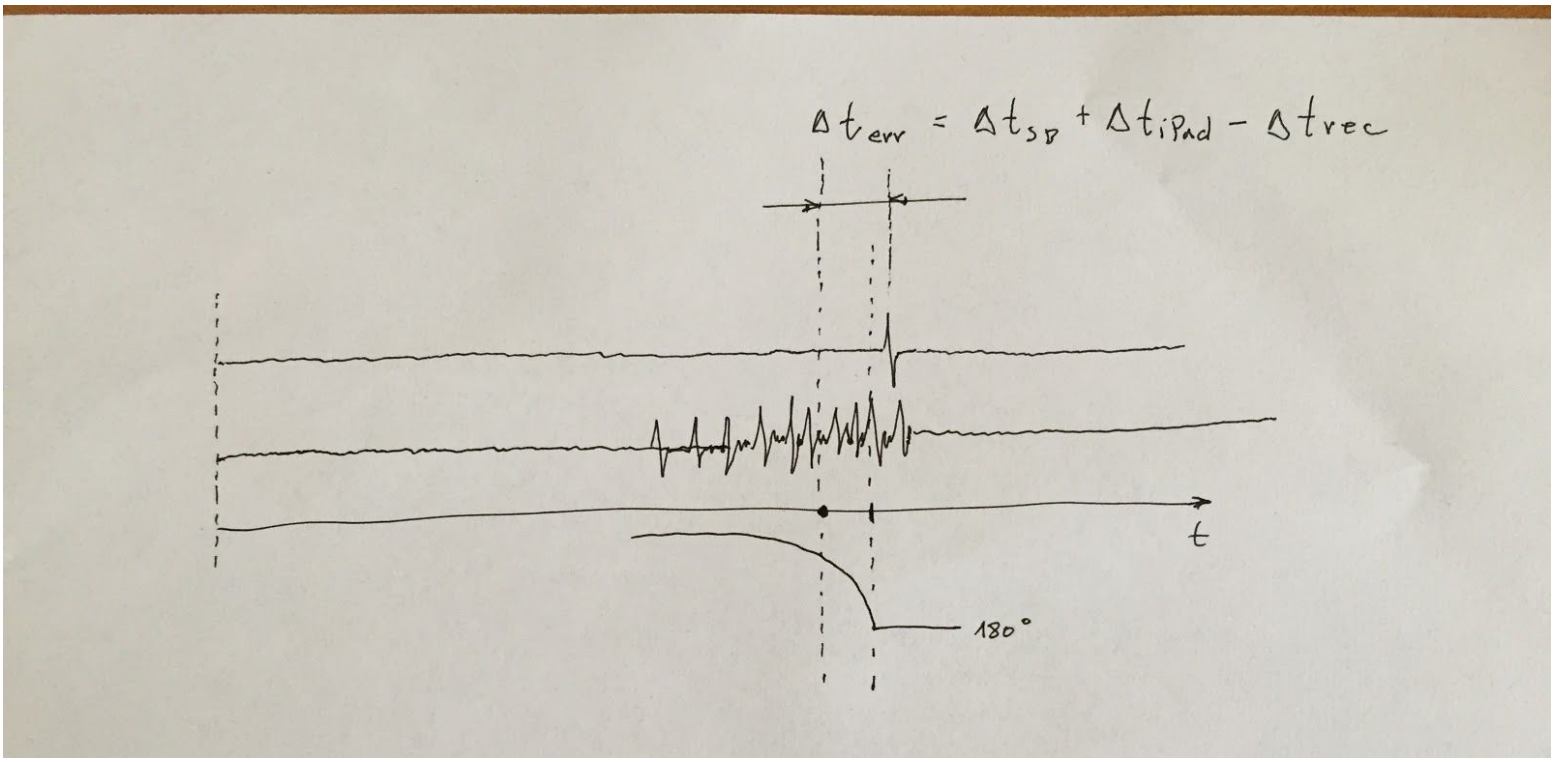


Image 4

Solution for this problem is to somehow measure Δt_{err} and to subtract it from our virtual time before we calculate our analysis/graphs.

If we can calculate exact moment (expressed in our graphical virtual time) when our circle object will fill iPad screen completely and if we can mark that same moment in recording we can subtract time difference between those two event and that time difference will be equal to Δt_{err} .

After that we can subtract Δt_{err} from virtual time and basically synchronize virtual time and recording. To be concrete, we can subtract that time from JSON timestamps for angles.

First, take a look at Image 5 where I calculated exact moment when circle will fill the whole screen. Important formula is:

$$t_{imp} = t_0 + \frac{dis \cdot \frac{S}{S_i} - d_0}{V}$$

Where t_{imp} is exact moment when circle will fill the whole screen expressed in virtual graphical time

t_0 - is moment at which we first time show the circle (first timestamp for angle in JSON)

dis - is distance from grasshopper to iPad (0.1m usually)

S - half the size of virtual object (if size parameter is set to 0.06 than this is 0.03m)

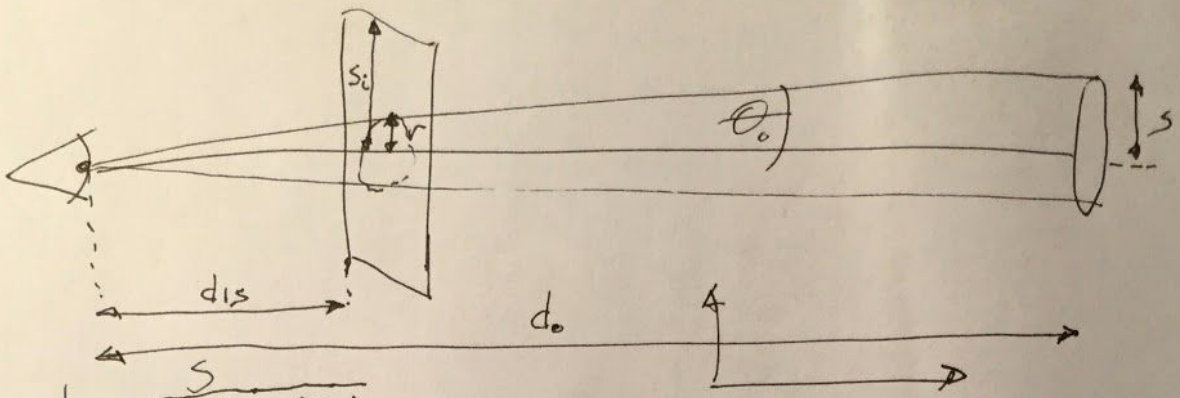
S_i - Half the size of iPad screen (we can use half the diagonal since iPad is 0.147m X 0.197m half the diagonal is 0.12290036615m)

V - is velocity (-2m/s for example)

d_0 - is initial virtual distance to object (how far is the object when we get first timestamp for angle in JSON). You can calculate that as:

$$d_0 = \frac{S}{tan(\Theta_0)}$$

Where S is the same half the size of virtual object (if size parameter is set to 0.06 than this is 0.03m) and Θ_0 is initial angle (first one in array) in JSON



$$d_o = \frac{s}{\tan(\theta_o)}$$

i.) $d = d_o + (t - t_o) \cdot V$ (velocity V is negative) — note

ii.) $s = d \cdot \tan(\theta) \Rightarrow \theta = \arctan\left(\frac{s}{d}\right)$

$r = dis \cdot tg(\theta) \Longrightarrow$ ⇓

size of object at screen → $r = dis \cdot \frac{s}{d}$

$s_i = dis \cdot \frac{s}{d}$ | ? d for iPad screen size when it will fill the screen

$d = dis \cdot \frac{s}{s_i}$

⇓ i.)

$d_o + (t - t_o) \cdot V = dis \cdot \frac{s}{s_i}$ | ? t

$t \cdot V - t_o \cdot V = dis \cdot \frac{s}{s_i} - d_o$

$t = \frac{dis \cdot \frac{s}{s_i} - d_o + t_o \cdot V}{V}$

time of impact when object fill the screen

$$t_{imp} = t_o + \frac{dis \cdot \frac{s}{s_i} - d_o}{V}$$

dis - distance iPad ↔ grasshopper
d_o - initial virtual distance grasshopper ↔ object
r - size/2 of object on iPad
s_i - half the iPad screen
θ_o - initial angle of half of the object
s - half the size of virtual object

Image 5

Second thing that we have to do is to mark in recording exact moment when circle fill the iPad screen. We can do that if we put phototransistor to one corner of the iPad screen and record output of that transistor like on Image 6. When screen in that corner goes from white to black we will have spike in recording that will mark the moment. (previously we checked if transistor introduces any substantial additional delay to system and we prove that it doesn't. Check previous emails that I sent)

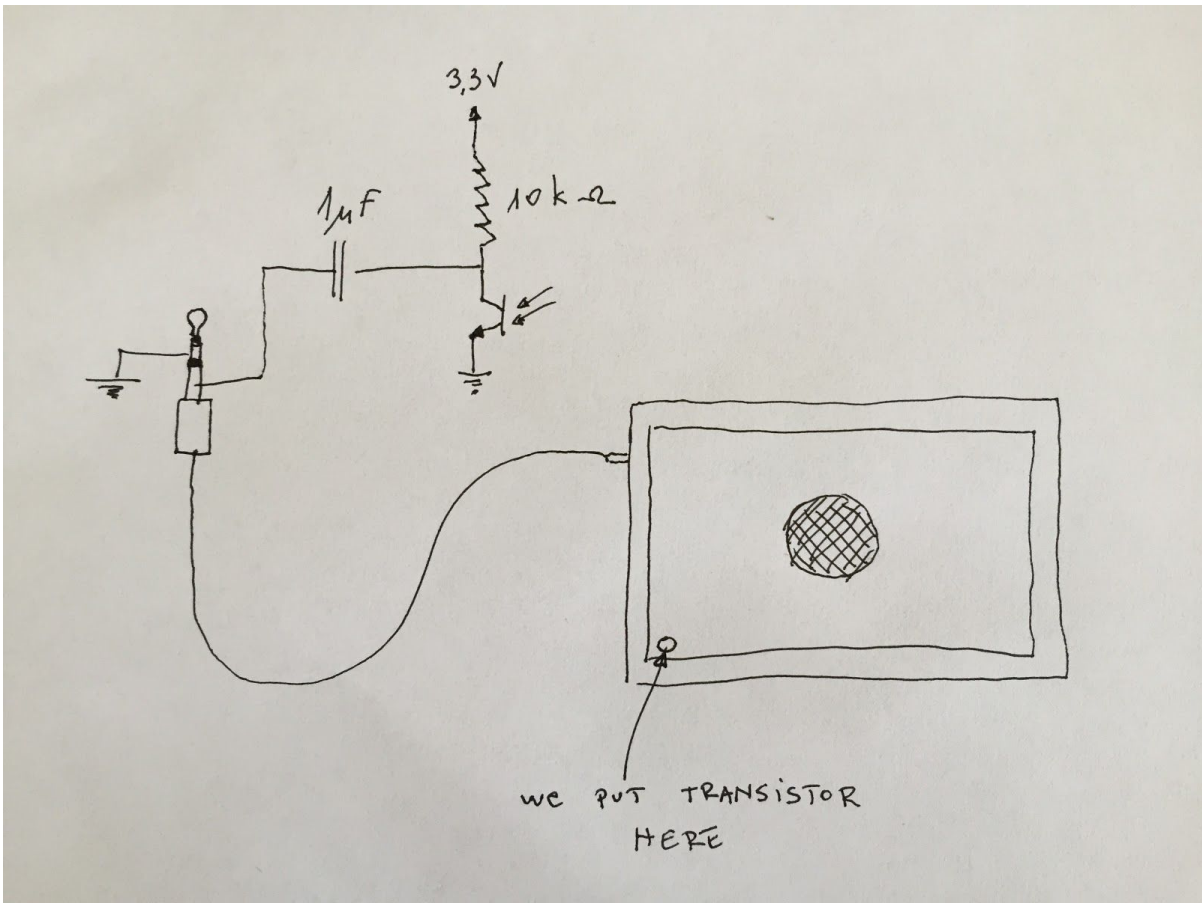


Image 6

When you make this experiment you will get recording like on Image 3 (signal named "Photo cell"). You can measure time of spike since beginning of the recording. We can name it t_{spike}.

Now you can get Δt_{err} as $\Delta t_{err} = t_{spike} - t_{imp}$. After that you can subtract cumulative delay Δt_{err} from all angle timestamps in JSON and do the correct analysis.