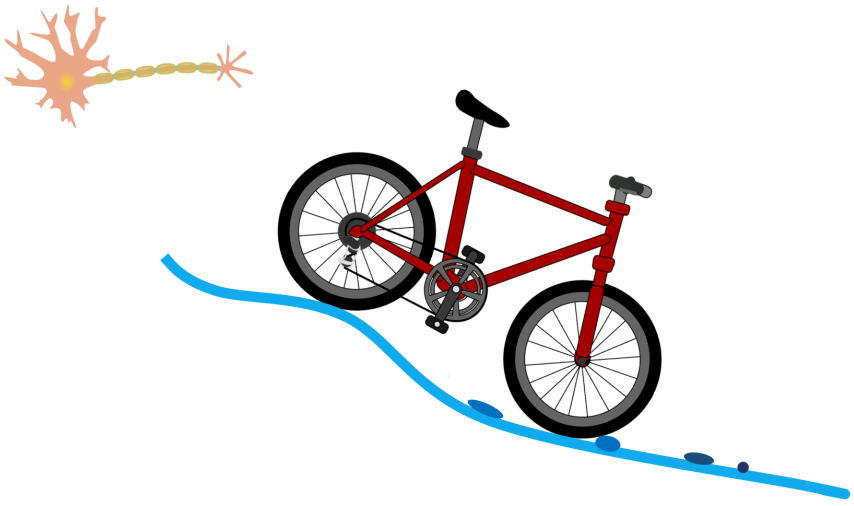
WheelCon Platform Manual

https://github.com/Doyle-Lab/WheelCon



By Quanying Liu & Ahkeel Mohideen

California Institute of Technology

Oct. 22, 2018

Content

1. Installation

1.1 Preparation

The gaming platform need be run in windows 10 (not in Mac). Please make sure that you followed these steps in your testing computer.

a) Install Driver for your Steering Wheel. We have tested the platform on Logitech G27 and G29 gaming wheel and the THRUSTMASTER gaming wheel. The other gaming wheels are not guaranteed to be compatible with WheelCon.

   If you are using the Logitech G27, please download the driver here.

   If you are using the Logitech G29, please download the driver here.

   The THRUSTMASTER driver can be download at this link.

b) Get Logitech Steering Wheel SDK with this link. It will be used for testing/calibrating your wheel.

c) Plug in Steering Wheel and run their SteeringWheelDemo.exe to see if sensors can be read and wheel has a force feedback. If it doesn't work, please install the DirectX 9.0c August 2009 SDK or try DirectX Update.

1.2 Installation of WheelCon

Download the WheelCon from GitHub: (https://github.com/Doyle-Lab/WheelCon).

Unzip it, and run 'WheelCon.exe' at your testing computer with the Steering Wheel connected.

Start with the Demo game, then try to specify your own game. See details about the Demo games in **Chapter 2**.

In **Chapter 3**, we also provide an example of feedback control model for our demo game. We also provide provide MATLAB code for the output file generated from the Demo game. You have to install MATLAB to run it. However, the output file is simply in the 'txt' format, and you can write your own code to analyze them in any platform, such as R, or Python.

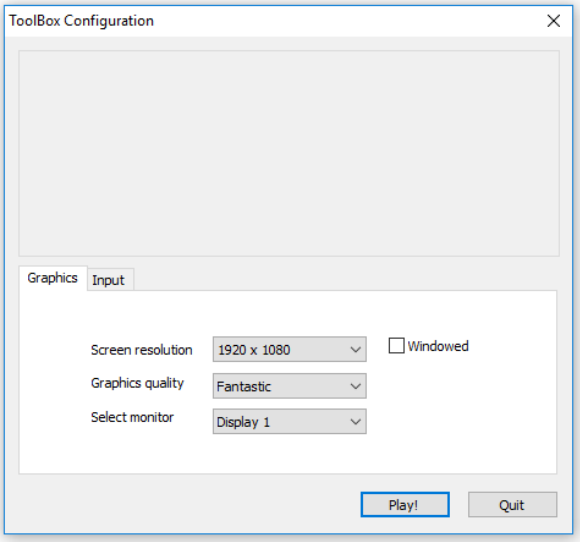
In **Chapter 4**, we show the lower-level development of new game using WheelCon. It is not needed to change the source code. You can simply generate new input files with specific format. We shared a simple example of MATLAB code to generate the input file for WheelCon.

In **Chapter 5**, we show the high-level development. You have to change the source code or write your own code. You need the basic knowledge of C# coding in Unity. Please install the visual studio for C# code and Unity for game development.

2. Playing the Demo games

We provide 6 demo games for you to start with.

Please start the platform by double click on 'WheelCon.exe'. You will see the following screen:



Unselect the 'Windowed', and choose the correct Screen resolution for your monitor. Click 'Play!'. You will see the main menu:

The Wheel Sensitivity (from 0 to 1) defines the sensitivity of wheel with low value meaning low sensitivity (you need a big angle of wheel to move the position).

Two tasks have been implemented in the platform: Fitts Law Task and Mountain Bike Task.

2.1 Fitt's law reaching game

When you choose the Fitts law game, it will go to the following gaming interface.

It mimics a traditional reaching task, which the player controls the steering wheel to the left / right, to reach a gray zone. The player should stay in the zone until the zone jumped to another area.

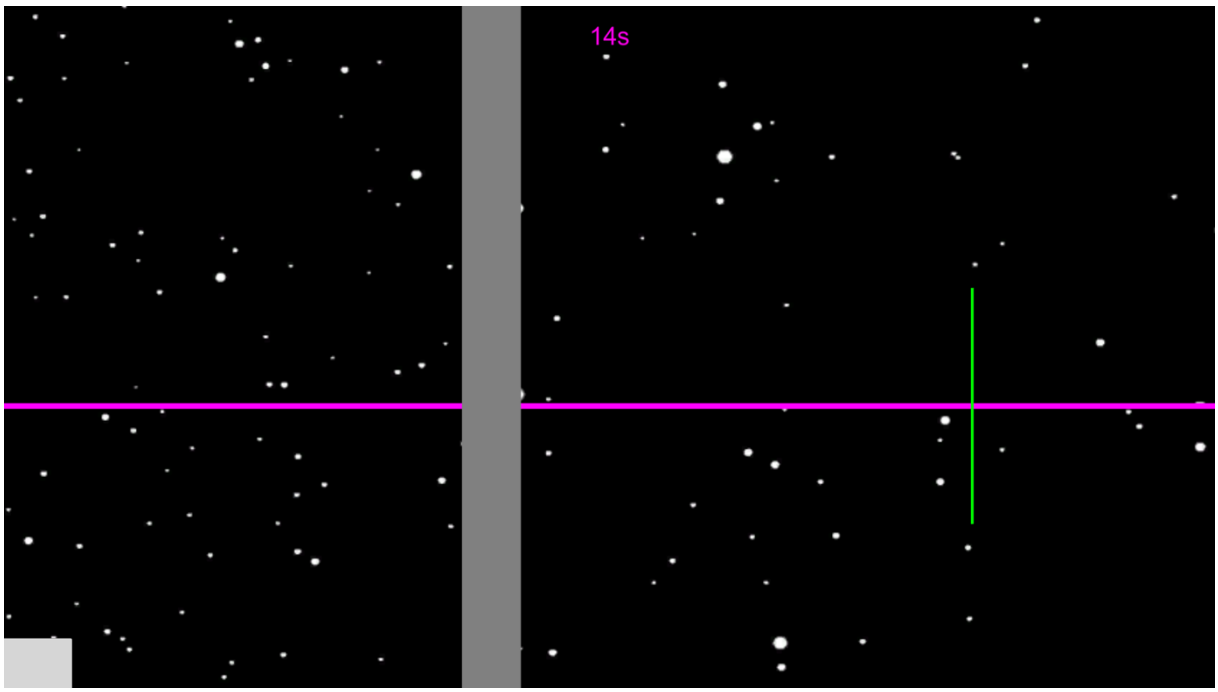
The Fitt's Law predicts that the time (T) required to rapidly move to a target area is a function of the ratio between the distance to the target and the width of the target. Let's define the distance to the center of the target (D) and the width of the target (W). T is a function of W and D:

$$T = p + q \times log_2(\frac{2 \times W}{D})$$

where p and q are two parameters

See more details about the Fitt's Law in Wikipedia.

Please type the output file name (for saving the output file), select 'Demo Input Files\t_path_fitts_law.txt', and click 'Begin Game' to start the Fitt's Law task. You will see the following gaming interface:



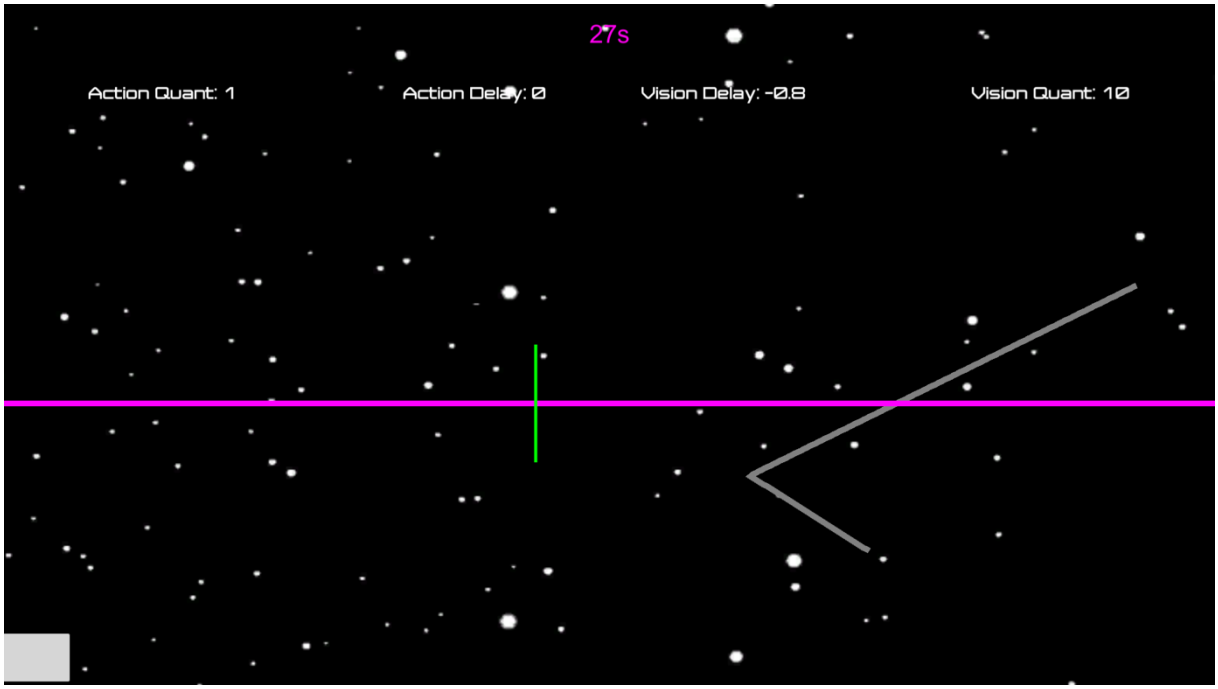
The gaming time is shown on top with purple text. The green line is your current position, and the grey zone is the desired zone to reach. The grey zone jumps each 2 second. When you see the gray zone in a new position, please steer the wheel to left or right to reach the zone and stay in the zone as quickly as possible.

The movement time is from the moment the grey zone jumping to another position to the time that the green bar staying in the grey zone.

2.2 Mountain bike task

When you choose ‘Mountain Bike Task’ in the main menu, it will go to the following gaming interface.

Please type the output file name, select 'Demo Input Files\setting_*.txt', and click 'Begin Game' to start the Mountain Bike task. You will see the following gaming interface:



The purple horizontal line indicates the present. The space above it indicates the future, and below it the past.

The grey line is the desired path, the green line is the player's current position. The task is to control the green line following the grey line with minimal error (the distance between the green line and grey line in the present time).

The quantization and delay for the visual input (negative delay means advance warning), and the quantization and delay for the action output are shown on top of the screen with white text. The gaming time is shown on top with purple text.

2.2.1 Vision delay and action delay

We start from the simplest game, with manipulating one parameter in the control loop: delay.

Please choose 'Demo Input Files\ setting_Vdelay_Adelay_worstcase.txt' to start the game.

In the game, we add either a visual delay or an action delay in the control loop. The visual delay is added via hiding the view of the trail. The action delay is added on the steering wheel.

We design a game with delay to test the impacts of delay. The sharp turns in the desired trajectory s(t) with the angle $\theta \in \{10, 20, \ldots, 80\}$ periodically arrive every 1.5 seconds. The disturbance r(t) = s(t+1)-s(t) with $\|r\|_\infty \leq 0.04$ unit (1 unit = 100 pixels). No advanced warning is added. Subjects

therefore cannot see any future trajectory. To examine the effects of vision delay, we vary the available history of trajectory in the visual feedback. At 0 delay, subjects cannot see the future trajectory, but they can see the full history of past trajectory. However, at 150ms, 300ms, 450ms, 600ms and 750ms of T_{vision} delay, they gradually lose the visual view of the past road. For the action delay test, an external delay is added to the steering wheel, but no vision delay. To be identical to the vision delay test, no advanced warning is provided. The 0ms action delay is the baseline setting; then 150ms, 300ms, 450ms, 600ms or 750ms of action delay, T_{action}, is added, all while keeping 0 vision delay so that the results are comparable between these two tests. Each setting in the game lasts for 30 seconds, so the total experiment is 6 minutes.

2.2.2 Speed-accuracy tradeoffs in the plan control loop

To test Speed-accuracy tradeoffs (SATs) in the higher-level plan layer, we design a game with visual advanced warning/delay and quantization. Please choose 'Demo Input Files\setting_SAT_worstcase_trail__ActQuant.txt' to start the game.

To test performance in the SATs with limited data rate (quantization) and advanced warning/time delay, we added either quantization in the actuator, or additional advanced warning/time delay in the visual input, or both. The quantization Q_{action} is put on the output from the steering wheel (the angle of the wheel), where the data rate, R_{action} is set to 1, 2, ..., 7 bits, respectively. The corresponding advanced warning/time delay, T_{vision}, is -800, -600, ..., 400ms, respectively. The negative value means advanced warning in the visual input, whereas the positive value means delay. For the T-R tradeoff settings, T = 200(R-5). Each setting lasted 30 seconds, so it takes 7*30*3=630 seconds in total (see Table 1). R_{vision} is 10 bits (maximum data rate for visual inputs) and T_{action} = 0 ms (no delay for the action output).

Table 1. Parameters setting in the SATs in the plan layer.

limited data rate	Gaming time (s)	0-30	30-60	60-90	90-120	120-150	150-180	180-210
	R_{action} (bits)	1	2	3	4	5	6	7
	T_{vision} (ms)	0	0	0	0	0	0	0
advanced warning/ delay	Gaming time (s)	210-240	240-270	270-300	300-330	330-360	360-390	390-420
	R_{action} (bits)	10	10	10	10	10	10	10
	T_{vision} (ms)	-800	-600	-400	-200	0	200	400
SATs	Gaming time (s)	420-450	450-480	480-510	510-540	540-570	570-600	600-630
	R_{action} (bits)	1	2	3	4	5	6	7
	T_{vision} (ms)	-800	-600	-400	-200	0	200	400

Please choose 'Demo Input Files\setting_SAT_worstcase_trail_VisQuant.txt' to start the game. In this game, we quantize the visual input. We only show the quantized gray line. The parameters are set the same as Table 1 but we vary R_{vision}. In this case, R_{action} is 10 bits (maximum data rate for action output) and T_{action} = 0 ms (no delay for the action output).

2.2.3 Speed-accuracy tradeoffs in the reflex control loop

To test SATs in the lower-level reflex layer, we design a game with bumps, and manipulate the delay and quantization in the action. Please choose 'Demo Input Files\ setting_SAT_Bump_worstcase.txt' to start the game.

The bump disturbance w(t) arrives every 2 seconds, with a maximum 100 units of torque. Similar to the setting for SATs in the advanced plan layer, we externally add either a quantizer with limited data rate R_{action}, or a delay T_{action}, or both in the steering wheel. The R_{action} is set to 1, 2, 3, 4 bits, respectively, and T_{action} is set to 0, 200, 400, 600ms, respectively. For the T-R tradeoff test, T = 200(R-1). Each setting lasted 30 seconds (in total 4*30*3=360 seconds). R_{vision} is 10 bits (maximum data rate for visual inputs) and T_{vision} = -1000 ms (800ms advance warning in the trail).

Table 2. Parameters setting in the SATs in the reflex layer.

limited data rate	Gaming time (s)	0-30	30-60	60-90	90-120
	R_{action} (bits)	1	2	3	4
	T_{action} (ms)	0	0	0	0
advanced warning/ delay	Gaming time (s)	120-150	150-180	180-210	210-240
	R_{action} (bits)	10	10	10	10
	T_{action} (ms)	0	200	400	600
SATs	Gaming time (s)	240-270	270-300	300-330	330-360
	R_{action} (bits)	1	2	3	4
	T_{action} (ms)	0	200	400	600

2.2.4 Bumps only, Trail only, and Bumps in the Trail

To be more realistic and natural, we developed a bump and trail dual-task game in the platform. Please choose 'Demo Input Files\ setting_Bump_Trail.txt' to start the game.

We either add bump disturbance w(t), or trail disturbance r(t), or bump and trail dual disturbance w(t) + r(t) in the game. To be noted, the w(t) and r(t) are independent. The bump effects were generated by a 0.5s constant torque to the steering wheel. We did not intentionally add external delays in the game (T_{action} = 0ms and T_{vision} = -1s), and we used the maximum data rate for both vision and action (R=10 bits).

Based on the feedback control model, we expect to see the sum of the error from the bump only task and from the trail only task is equivalent with the error from the bumps in the trail task. See more details about the model in the following chapter.

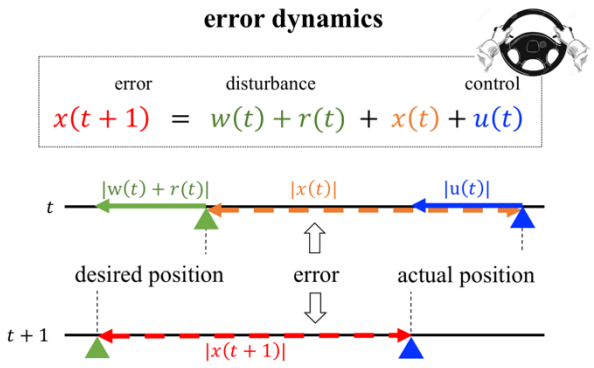
3. Model and data analysis

3.1 Feedback control model

We show a simplified feedback control model shown in Figure 4. The system dynamics is given by

$$x(t+1) = x(t) + w(t) + u(t) + r(t)$$

where x(t) is the error at time t, r(t) is the trail disturbance, w(t) is the bump disturbance, and u(t) is the control action.



3.1.1 Modelling action delay in trail disturbance

When there is a delay T in the action, and a trail disturbance r(t), we model the control action by

$$u(t+T) = \kappa\big(x(0:t), r(0:t), u(0:t+T-1)\big) .$$

The game starts with zero initial condition, i.e., x (0) = 0. The controller κ generates the control command u(t) using the full information on the histories of state, disturbance, and control input. The control command is executed with delay T ≥ 0.

Sensorimotor control in the risk-aware setting motivates the use of L1 optimal control, and as such, our goal is to verify the following robust control problem

$$\inf_{\kappa} \sup_{\|r\|_\infty \le 1} \|x\|_\infty .$$

This problem admits a simple and intuitive solution. The optimal cost is given by

$$\inf_{\kappa} \sup_{\|r\|_\infty \le 1} \|x\|_\infty = T .$$

This optimal cost is achieved by the worst-case control policy $u(t+T) = -r(t)$, which yields

$$\inf_{\kappa} \sup_{\|r\|_\infty \le 1} \|u\|_\infty = 1 .$$

3.1.2 Modelling action delay and quantization in trail disturbance

In the 'SATs in the plan control loop' game, the control action is generated by the following feedback loop with communication constrains,

$$u(t+T) = Q\big(\kappa_t\big(x(0:t), u(0:t-1)\big)\big) ,$$

where $\kappa_t : (\mathbb{R}^{t+1}, \mathbb{R}^t) \to \mathbb{R}$ is a controller, and $Q: \mathbb{R} \to S$ is a quantizer with data rate R ≥ 1, i.e. S is a finite set of cardinality 2^R. Here, the net delay is composed of the internal delays in the human sensorimotor feedback and the delays externally added. The disturbance r(t) is infinity-norm bound and without loss of generality, $\|r\|_\infty \le 1$.

The worst-case state deviation is lower-bounded by

$$\sup_{\|r\|_\infty \le 1} \|x\|_\infty \ge \mathrm{T} + \frac{1}{2^R - 1} \ ,$$

and the minimum control effort is given by

$$\sup_{\|r\|_\infty \le 1} \|u\|_\infty \ge \left(1 + \frac{1}{2^R - 1}\right)\left(1 - \frac{1}{2^R}\right) \ .$$

3.2 Data analysis

To analyze the data, we used MATLAB code. Please install MATLAB and run the demo 'Demo Input Files\setting_SAT_worstcase_trail__ActQuant.txt'. You will get an output file.

Please open 'Source Code\WheelCon_code_SAT_plan.m' with matlab, and adjust the variable 'folder' and 'file_names' to your folder and output filename.

Run the code, it will generate a figure with speed and accuracy tradeoffs in the performance.

Specifically, the raw data will be in the following variables.

```
M = dlmread(file_names,',',1,0);


% load the data
t = M(:,1);
trail = M(:,2);

bump = M(:,3);
quant_act = M(:,4);
delay_act = M(:,5);
delay_vis = M(:,6);
quant_vis = M(:,7);

error = M(:,8);
control = M(:,9);
```

This code if only for an example to analyze the txt output file in 'SATs in plan control loop task'. Feel free to write your own code to analyze the data.

4. Low-level development

For the low-level development, you only need write your own the input file. It is not necessary to know the source code with C# and unity.

The Input file needs to be setup in a specific way in order to get accurate experimenting. The file should be stored as a “.txt” file and all the information will be written in text. Each experimental variable is represented as an animation curve in the software. This means that at every time every experimental variable must be associated with a value. The resolution for the time is at minimum 10ms (0.01 seconds). Therefore, in the input file, all experimental variables must be valued at every time stamp, every 10 ms (or the desired time resolution). Each line of the text file should represent a single time stamp with appropriate variable values. The format of the line depends on the version of the game being played.

For the mountain task the format is:

Time, horizontal position of the trail, bump size, action data rate, action delay, vision delay, vision quantization

e.g. 0.01,6,10,-1,30,0.2

Please note the lack of spaces.

Each line will be read by the program, interpreted, and the necessary information inputted into their respective animation curves. The time is used as the input to the animation curve. There is no need for a line to signify the end of the game. The experiment will end as soon as the game time has exceeded the last declared time stamp in the input file. If the time in the last line of the input file reads 39.90, the game will end at 40 seconds.

For the experimental value, it is best to look at an example. Take angle of the trail. The angle of the trail ought to be set at every time stamp. In the example above, the trail will be at an angle of 6 degrees when the current game time is 0.01 seconds. This means to create a straight line the angle of the trail value should stay constant for as long as the experimenter wishes the line to be straight and continue in the same direction. If the experimenter wants a straight line at an angle of 6 degrees for 10 seconds, there ought to be 1000 lines (assuming a time resolution of 10 ms) in the input file that read as such: time,6,⋯

All the other experimental values will work in this same manner. This is to say, do not expect the software to hold a certain experimental value constant until a change in value is desired or declared. The value of each and every experimental variable must be declared at every instant the game is in play. If no value is provided, the software will default to 0. If a value of 0 is desired, it is best to explicitly state it in the input file as ⋯,0,⋯

The horizontal position of the trail value represents the x position in the screen for the grey line.

The quantization level corresponds to the number of bits allowed for communication between wheel and computer. The wheel has a very fine resolution when measuring its current angle. It is worth noting that although the wheel can be rotated many times over, there is only 240 degrees of accepted angle range for the game's purposes. Adding quantization means that the wheel's angle is quantized into discrete regions. At a quantization level of 1, only 1 bit is allowed for communication between the wheel and computer. This means that there are only 21 angle values. In other words, all 240 degrees of movement are quantized to 2 possible angle values. These lie at the extremes of either maximum angle to the right (120 degrees) or maximum angle to the left (-120 degrees). With a quantization level of 1, if the wheel's current angle is less than or equal to 0, the software moves the player's position as if the wheel was at -120 degrees. If the wheel is at any angle greater than 0, the software moves the player as if the wheel was at 120 degrees. The reason the player is not allowed to go straight is because that would mean the player was capable of moving at -120, 0, or 120 degrees. This would be more than 2 options and so not capable of communicating with only 1 bit. With higher levels of quantization, player motion near perfectly reflects the angle of the wheel. There is a minimum level of quantization which is 1. However, the way the software is written, there is technically no maximum level of quantization. The quantization level ought to be an integer value greater than 0. The software will run if decimal values are used but the number of quantization regions will be truncated to an integer within the software.

e.g.
quantization level of 2 → $2^2 = 4$ regions of motion → 240 degrees of motion is divided into 4 regions

quantization level of 3.5 → $2^{3.5} = 11.31$ regions of motion → 240 degrees of motion is divided into 11 regions (truncated)

5. High-level development

Wait for AM.