Motion Controlled Multiplayer Pong

Datorteknik, IS1200

October 10, 2025

Project Members:
[Razmus Nilsson, Razmus@kth.se, 990817-3439]
[Kai Nguyen, kaingu@kth.se, 041006-8357]

Objective and Requirements

With this project we have developed our own version of the classic 1972 game "Pong". In this game, two players compete to score points by ping-ponging a ball back and forth, attempting to launch the ball past their opponent's pad and hitting their goal (wall). First to score 10 points wins. What makes our version a bit different is that we have used two accelerometers as input to allow for motion control of the paddles, which enables an active and engaging multiplayer experience. The two players will control the pads in the Y-axis to defend their respective goals.

The following are the requirements we have met from our original draft to our final project:

- Multiplayer (2-player) and motion control are the main focus of the game. The users can control their respective paddles using an external accelerometer.
- The game features and keeps track of the score to reach a win-condition for one of the players. The winning player is displayed at the end.
- The ball reacts to where on the paddle it hits and changes its trajectory accordingly.
- The entire game is programmed in C.

Following are the original objectives that were omitted from this project:

- The original plan for the project and a big point was to use randomized power-ups that would spawn into the game and change up the gameplay.
- The game only controls in the linear Y-axis instead of both Y- and X-axis.
- We intended to use a bitmap to display the graphics.

Solution

The entire project works exclusively on our provided DTEK-V board, being developed in C. Our two accelerometers of the type MPU6050, act as gyro-sensors. The MPU6050 gives readings on acceleration in the x-, y- and z-axis. For our project, we only read the acceleration for the y-axis. This acceleration value gives us how much distance we need to move the paddle in the game engine. The MPU6050 is connected to a breadboard which is then directly coupled with the DTEK-V board. To allow multiplayer, we just need to read the input from both MPU6050s simultaneously. To allow the accelerometers to function properly we had to use an external library dtekv-i2c-lib[1]. This enables us to properly communicate with I2C devices (our accelerometers) over the DTEK-V's GPIO pins without the use of an external I2C bus, which was the original plan for this project. The game is displayed from the DTEK-V board's VGA output into an external compatible display where the graphical interface can be seen.

Verification

To verify that our game works properly as intended we have done multiple and mostly repetitive testing of the game's behaviour. Following will categorise these different cases and how we've handled them:

The core gameplay starts in the main function by drawing and initializing our objects and sensor. It then enters the gameplay loop when button one on the DTEK-V board is pressed. the function `handle_interrupt` will keep the gameplay-loop active until the win condition is met, while it keeps track of updating the graphics and handling that the paddle and ball positioning remains active. Inside of the function `start` the accelerometers will update the current position of the paddles.

An important part of Pong is the ball's behaviour and physics when interacting with the paddles or hitting the walls. Since we don't want the ball to reach out of bounds, we've defined and simply invert the direction of the ball when it hits either the upper or lower bounds, same way the paddles are not able to move out of bounds in their y-axis. The ball's trajectory when bouncing off the paddle is decided by a self defined sine (movement in X-axis) and cosine (movement in Y-axis) approximation, using the Taylor series definition. The max bounce angle is defined by a PI constant / 4, which gives us a max bounce angle of 45°.

When the ball hits the paddle the angle is calculated by taking the difference between the middle of the paddle and the ball's hit position on the paddle. The ball's hit position is then normalized between -1 and 1 which gives us a bounce angle when `hit_position` is multiplied by `max_bounce_angle`. This bounce angle is then taken through our cosine and sinus functions to calculate the new movement in the X- and Y-axis respectively.

The defining feature of our version is the use of external accelerometers. This in addition with adding a second one is as previously mentioned only possible with the external library. Since the library is only defined for a single I2C device, we had to edit it to read from both of MPU6050 devices separately, we read constantly the input from both during the gameplay loop.

To convey movement inside we game we do a simple act of drawing, erasing, and re-drawing the objects, in their new positions if they've moved. Since we're constantly doing this act of erasing and re-drawing to keep track of the movement all objects on the screen has a constant flickering visual. Both the score and the "net" is re-drawn constantly as well due to the ball passing over these would act as a eraser if we omitted them from the constant re-drawing.

Lastly we have done testing that the score updates and is displayed accordingly, which is whenever the goal (wall) of each respective player is hit. We exit the gameplay loop as soon as the winning condition of 10 is reached. The score and winning message both use 2D array-bitmaps to store the characters. The way we draw characters is to draw each row individually. Each element in the first dimension represents a row of 5 pixels (5 bites) wide and in binary is either 1 or 0 (on or off). All elements are 7 pixels in height. We take a string of characters as input and then translate it to pixels with our functions `draw_text` and `draw_char`.

Contributions

Since we chose to omit certain part of the final project the distribution of work shifted during development. Following is how the final work-load was distributed between the two of us.

Kai handled:

- The accelerometer's implementation and behaviour. He handled all external equipment and how the logic would function inside of the game. As well as the button logic to start and restart the game.
- The physics of the ball and its behaviour, especially its trajectory when bouncing of the paddle or the upper and lower bounds.

Ramzus handled:

- The graphics inside of the game and how it updates consistently under the gameplay loop.
- The implementation of the score and the display of text when the winning condition is reached.
- The scoring logic and the winning condition.

Reflections

In our first abstract draft we included certain objectives that have been omitted from the final project. Mainly being the power-up feature that was supposed to give the player that collected it a small temporary advantage. Unfortunately due to time constraints (and being more challenging than originally thought to implement) we chose to remove this feature, since it was not necessary for the core gameplay-loop.

We also wanted to display our objects in game with the use of a bitmap to draw each pixel individually on the screen, this was also omitted due to it simply not displaying properly and also being more difficult to implement than the option we chose to proceed with. We instead chose to display our graphics as blocks of pixels instead, which worked just as well. We did however use two separate bitmaps to display our score and winning message. Since these were only for display and did not have any physics, it worked properly without hassle.

References

1. Olivoz. *dtekv-i2c-lib*, 2025. Available at: `https://github.com/Olivoz/dtekv-i2c-lib`