# Journals

## Exercise 1 and 2

For the first two exercises, we just followed the guidelines, and got up and running fast. While we were doing the first couple of exercises, we had chosen to use HyperTerminal to display our output. However, after discussing this with a couple of other groups, and playing with Putty ourselves, we chose to change our terminal over to Putty. Here we have more options to customize the output from the controller. As we did this, we put some thought in to setting as much up as possible through ANSII escape commands, as it would look somewhat similar in the two terminals.

We had a few issues trying to print our first escape commands, as we were not sure what to print. First, we just tried to print 27, the ANSII number for escape, as a decimal, which just resulted in printing out the number. After we discovered the number just needed to output as a char, not decimal, everything worked as expected.

As we started working on the window in the terminal, we tried to separate as much as possible into another file, which we then later could use. We have called this ansi.c and ansi.h, which is not a very good name, but then we knew which project it was associated with. In these files, we had all the escape commands we thought we would use later in the course, as well as the function to create a window in the terminal with a given size and text.

## Exercise 3 and 4

When we had created the sinus and cosine functions, we had trouble getting the exact results we had given in the exercise guide. After trying different options to get different entries into the lookup table, we determined that our original functions gave is the best possible values for each angle. We then compared the error margin produced by our functions compared to those given in the exercise guide, and saw that we had a marginally small difference, but that ours were always a bit more off. However, this was not something we could do much about, without writing our own lookup table. We decided to let the code take care of the conversion of the angle into the lookup value, as we did not see that as a too heavy calculation, compared to manually calculating it each time. The cosine function just works by calling the sinus function, and adding 90 degrees.

The rotate function was just to implement the formulas. This did not give us any trouble, except for our values differ compared to the values on the exercise guide, but that was due to the sinus and cosine functions. We played a bit around with the different fixed-point formats, starting with just calculating in 16.16 and then realizing we had to use the 18.14 format. In the code we use for our product, we those to use a 24.8 format, which was done to insure we did not lose any data. However, we also lose some precision by doing this, but this was acceptable due to our later conversion to integers.

## Exercise 5 and 6

While doing exercise 5, we experienced a lot noise when reading on our input signal. This meant we had a hard time reading a single button press. To counter this, we used a timer, to wait between each button press. By doing this, we were able to register each press individually, while the user would not experience any lag. When we sat up the displays to show a binary counter, we also started building on our display library. In this, we made different functions to clock each display or all of them at once. When we then wanted to display the counter as a binary number, we could just set the output to the char where we stored the number. This way, we could count up to 127 on our display.

Because we already played with the timer here, and had made a library of the functions we used, we thought we had the necessary experience with timers, and therefore skipped the most of the sixth exercise. We did a quick project, where we calculated the time between to presses, but did not go deeper into this exercise.

## Exercise 7 and 8

We started by setting up the video buffer, as described in the exercise guide, and then started to copy over different sections form the given charset. To make our displays show text, we used our library from the previous exercise. We had previously created a function to set the output to a given char, and by looping through our video buffer, we could pass different entries in the array at different time, and clock the correct display to match data. This way, we could with a simple loop display anything saved in the video buffer.

We did not do the last exercise, were we should have made the text scroll along the displays. Our thought to solve this problem, was just to have a pointer at the start of the video buffer array, and then move it along when scrolling. This would save us from coping all the data each time. This way, we could also just reset the pointer to the start of the video buffer, when we got to the end, and thereby continuing the scrolling.