Position Ramping Curve

Problem Statement

Your task is to write code in C that, when invoked, transitions positions from the current

position (a.k.a. “Starting Position”) to the desired weight (a.k.a “end position”) by making many incremental position changes. These positions changes can be assumed to take place instantaneously.

The end result will be a trapezoidal position ramping profile. See the section labeled “Trapezoidal move profile”:

https://www.linearmotiontips.com/how-to-calculate-velocity/

Your trapezoidal position ramp should aim to complete the transition to the end position as quickly as possible while not violating the bounds as listed in the requirements below.

Requirements

1. All work should be original - we love to reuse open source work but we would like to review original works for this problem.
2. When changing position upwards (i.e. increasing position), the maximum ramp rate is 130 mm/s^2.
3. When changing position downwards (i.e. decreasing position), the maximum ramp rate is -80 mm/s^2.
4. 1/6 of the total position should be ramped in the first phase of the trapezoid
5. 2/3 of the total position should be ramped in the second phase of the trapezoid
6. 1/6 of the total position should be ramped in the third phase of the trapezoid
7. The ramp profile file(s) should be structured in a way that it would be easily ported to a bare-metal embedded processor
8. Architect your code in a way that the ramping profile can be run multiple times
1. concurrently - (assume there are multiple systems that need to utilize this code that may run in parallel)
9. Assume that the calculation of your ramping curve is called from an ISR every 20 ms.
10. The input position is a variable that can be easily changed anywhere from 50 to 100 mm in different ramp contexts

Deliverables

When you have completed your solution, please provide:

1. Your implementation and all related sources
2. A list of any assumptions you made

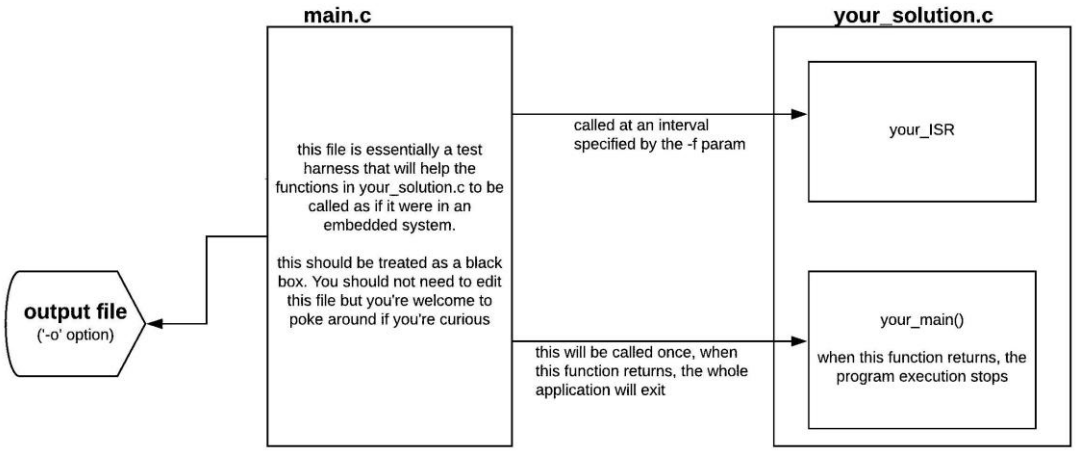
3. A list of the parameters used to test your final solution
4. Posix based project including the ramping profile, build instructions and test harness.
5. Sample .csv for a 50-100 mm ramp with each row including the timestamp, total weight, and incremental weight applied during that time-step.
6. A image of a graph showing the sample .csv from the 50-100 mm ramp - see the provided plot_weight.py for help (more documentation below)
7. Be prepared to discuss architectural decisions that were made as well as design tradeoffs around latency, resource usage, maintainability, etc.

Hardware Emulation

The rest of this document is aimed at getting you comfortable with the emulation software (a.k.a. “Test harness”) that has been provided with this problem statement. Refer to it as needed.

Hardware Emulation Test Harness

In order to aid you in this exercise, we have provided a testing harness that emulates the hardware available in an embedded system. There are two relevant files, main.c and your_solution.c. The main.c file that is provided manages the hardware emulation and should not need to be edited or even read unless you are curious. A makefile has also been provided that will build and link all .c files in the working directory. It was intended that your work be done in your_solution.c



The following command line options are available in the test harness:

Option	Meaning	Required	Notes
-o	Output file name	yes	This file will be a .csv containing “time (ms), weight (lb)” pairs needed to plot with the included plot_weight.py. The contents of this file are generated when calls to apply_position() are made.

			When the program launches, the starting weight is automatically logged
-s	Starting position	yes	This is the starting weight or current weight. It can be anywhere from 5 to 100 lbs.
-e	Ending position	yes	This is the ending weight or target weight. It can be anywhere from 5 to 100 lbs.
-f	Delay between calls to YOUR_ISR()	yes	Takes as a parameter a millisecond delay. You should pick this value to reasonably match the accel sample rate. Don't worry about jitter, etc.
-d	Debug option	no	If this flag is provided, the print_debug_log() function will cause the incoming strings to be printed like 'printf()'. If this flag is not provided the print_debug_log() will have no effect The get_is_debug() function will return this value as a boolean, true if set, false otherwise

Your_Solution.c

You are welcome to create additional files as and if needed, but it is intended that your solution would live in the file your_solution.c that is provided. You will find 2 functions in your_solution.c that are important to this question:

```
/*
 * This function is the equivalent of a typical main() function.
 * This thread will run at a lower priority than YOUR_ISR().
 * When this function returns, the harness will cause program execution to end immediately.
 */
void your_main(void);
```

```
/*
 * This function simulates an on-target ISR. it is called at a regular interval
 * at a higher priority than the your_main() function.
 *
 * The rate at which this function is called by the test harness can be set
 * with the '-f' option. the '-f' option takes the time between calls in
 * milliseconds as the parameter
 */
void YOUR_ISR(void);
```

Helper & Emulator functions:

We have provided the following function declarations for you to use during development in util.h, they exist to help you and you can choose to use them or not.

```c
/*******************************************************************************
 * here are some convenience function to aid you in your development. you
 * do not need to utilize the functions if they are not helpful for you
 *******************************************************************************/

/* convenience function. if the debug flag is set, this will become a call to
 * printf. if the debug flag is not set, this call will be ignored */
int print_debug_log(const char *format, ...);

/* set the position the motor is currently generating. this position will be applied
 * instantaneously. it is up to you to make subsequent calls to this function
 * to get a position ramp that follows the bounds of the question - instead of
 * applying the position, this emulation hardware just logs this to the output
 * file provided in order to generate a plot */
void apply_position(float position_mm);

/* get the starting position or current position as set by the command line option */
uint8_t get_start_position(void);

/* get the ending or desired position as set by the command line option */
uint8_t get_end_position(void);

/* this option is false unless the -d flag is set on the cmd line
 * (i.e. "./solution -d") */
bool get_is_debug(void);

/* get the current system time in us */
uint64_t current_timestamp_us(void);
```

A note about control systems in this problem

For those familiar with control systems, you can consider calls to apply_position() to be what sets our reference trajectory in our underlying controller. If this is not helpful or does not hold meaning for you please feel free to ignore.

Using The Emulator

This section is here purely to help you get acclimated to this question more quickly.

Software Needed

The emulator tested on Ubuntu 22.04 LTS

>> make –version

GNU Make 4.3

>> gcc –version

gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0

Running the Emulator

In order to build all .c sources into your solution, you can run make against the provided makefile:

>> make

After building, you can run the sample code we have provided at a 20 Hz ISR frequency in debug mode by running the following command. The starting position is 50 mm. The ending position is 100 mm and we're generating test.csv.

>> ./solution -f 20 -o test.csv -s 50 -e 100 -d

You will see the test.csv file now contains time (ms) and position. Note that the starting position is automatically added as the first row.

```
>>cat test.csv
0.000000, 5.00
1001.312000, 35.00
2002.486000, 95.00
```

Plotting the Result

Now that you have run your weight ramp simulation, you can now run the python plotting script on the output file. Remember, each row in the output file is generated from a single call to the apply_position() function:

>> python3 plot_position.py -f test.csv

The starting code will generate the following plot if untouched: