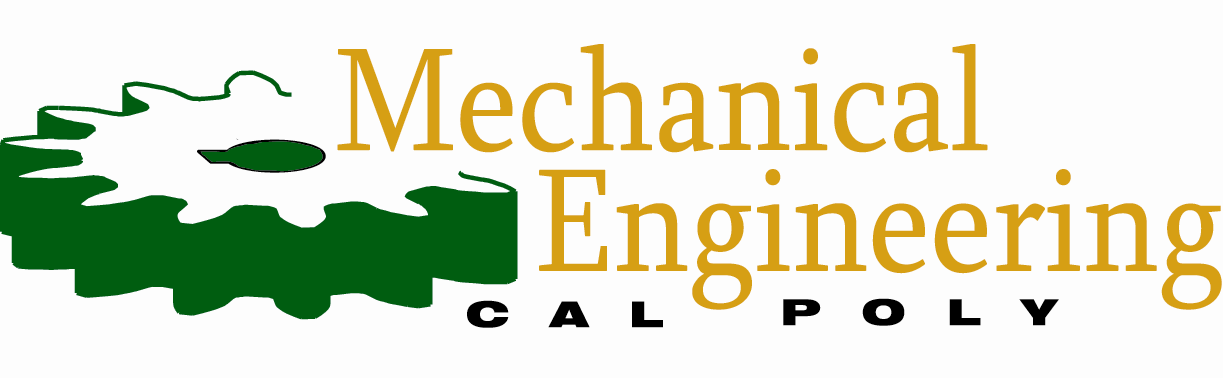
**MEMORANDUM**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **To:** | Kennedy, Jane E. | |
|  | jekenned@calpoly.edu | |
| **From:** | Edward Ruano | Robert Trujillo |
|  | eruano@calpoly.edu | rotrujil@calpoly.edu |
|  | Drew Isaak | Anthony Lombardi |
|  | aisaak@calpoly.edu | atlombar@calpoly.edu |
| **Date:** | April 28th, 2016 | |
| **RE:** | **Lab 3: The Need for Speed** | |
|  |  | |

In this lab we were tasked with creating a driver, task and two interrupt service routines for our encoder. We were also required to write a test program to exercise our driver to allow for debugging. The completion of this lab will let us accurately monitor the speed of our motor so that we can calculate positional data like speed and displacement.

Our driver sets up the interrupts and allows our code to access the counter which holds the encoder's current position. It also is able to detect errors that occur such as missed pulses. We are able to access the shared variables that contain the encoder count and state, which allows us to set initial value or set to zero.

The task\_encoder class takes a pointer to an encoder\_driver that is initialized in the main() function. The ISR code resides within the encoder\_driver class so the task\_encoder's main objective that runs within its run() method is to compare the current value of the encoder\_count shared variable to its previous reading every 10 milliseconds. The bulk of our ISR logic is based the encoder\_driver class.

One of the interrupts we created responds to two external interrupt sources and can come from changes in the encoder's output. The interrupt first stores the previous state and then gets the current reading from pins A and B. From there our logic is able to determine the direction based on the two bit reading from both pins and adds or subtracts one from the shared variable encoder\_count. The second interrupt is similar to the first interrupt but allows us to reduce our logic in each of the interrupts, resulting in each of them working faster.

For testing we monitored our error while under normal motor conditions and unconventional conditions. We used a button to test interference by sending multiple signals to the interrupt pins which in theory would set them high or low at inappropriate times. The code operated well when the encoder was reading 15 tick per millisecond or less but had issues at higher speeds because of the high resolution of the encoder. To operate at higher speeds we will be using an encoder counter chip in the future.