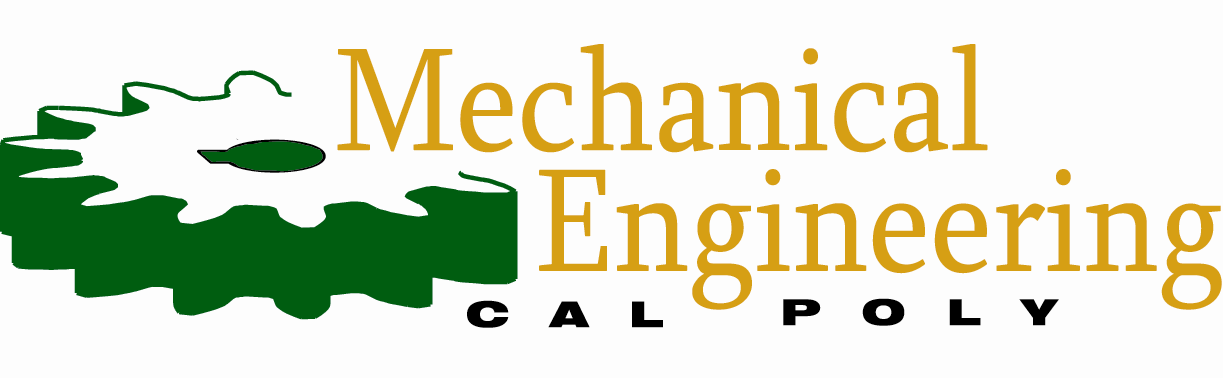
**MEMORANDUM**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **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:** | May 5th, 2016 | |
| **RE:** | **Lab 4: PID Controller implementation** | |
|  |  | |

The controller we implemented uses proportional, derivative and integral operations on the feedback we route into it. The PID uses a motor set point value given to it by the task\_user and therefore the user since we do not have active communication set up yet. This set point is a value from -1024 to 1024 with negative values corresponding to reverse velocity.

The feedback we chose to use was the ticks counted on the optical encoder mounted to the motors' rotating shaft. We calculated one motor shaft revolution to be 1800 encoder ticks, with one motor shaft revolution also corresponding to 1/64 rotation of the rear wheels in low gear and a 1/31 ratio in high gear. With one wheel having a circumference of 26 cm, we were able to calculate max speed (in low gear) to be around 34000 ticks per sec or 3.83 cm per sec or 0.04 meters per sec.

When at this high speed we obtained a moderately stable encoder feedback value of about 34 ticks every ms. Since the task\_encoder is owner of the highest scheduler priority and there is a hard 1 ms run delay declared, we can have confidence that the value is stable and most likely being recorded every 1 ms.

Furthermore, we are using an external counter chip for the optical encoder because the number of interrupts was to large for the 1281 chip to handle. Every 1ms, this chip is queried and the value extracted and compared to the previous value stored in the task memory. Thankfully, at full speed, about amounts to about 46 % of the total ticks on the encoder wheel so the task\_encoder always knows whether we're changed direction since we catch the next value before the overflow makes it uncertain. This value is then updated into the encoder\_ticks\_per\_ms variable in the shares.h file.

Using this value, the PID performs proportional, derivative and integral control operations to try and get out out set point to reflect onto the motor. The values we decided to give the PID were values of 1, that gave us timely but smooth transitions between our motor set points. We included a graph below of a the response we obtained from applying a motor set point of 1024 over the course of several RTOS ticks which are defined to be close to 1 microsecond by the Ridgely's comments in the RTOS and ME405 libraries. We made sure to keep this PID generic enough to use with our vehicle's other functions like steering and perhaps integrating the shift to get to unreachable velocities if held in a constant low gear.

\*\*\*\*\* INSERT GRAPH BELOW