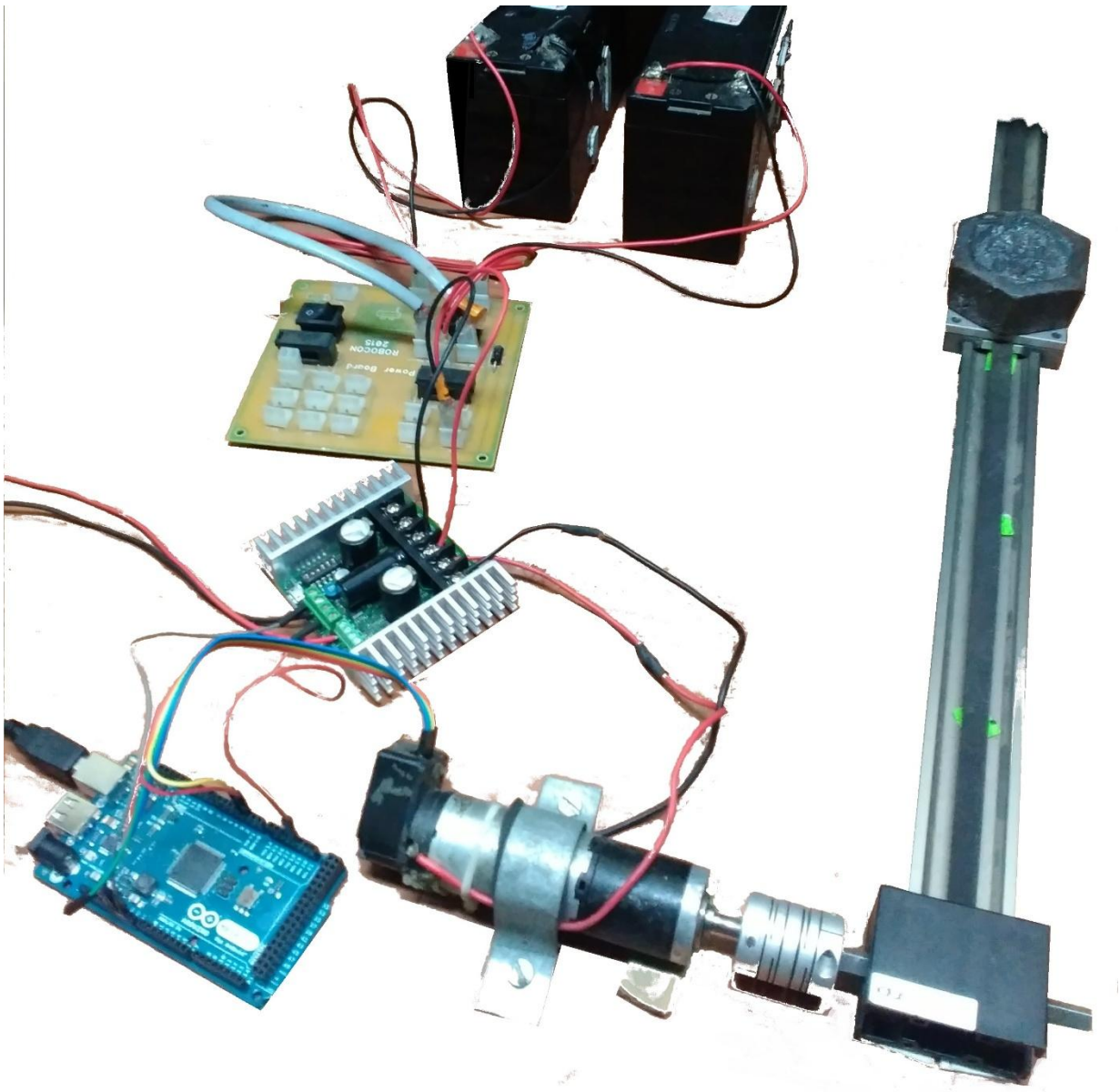


SLIDER CONTROL

SUMMER PROJECT REPORT 2015

Project deals with precise positioning of slider with the help of ramping.

TABLE OF CONTENTS

Contents

Aim

BRIEF DESCRIPTION

Robots, in order to gain the freedom of motion, use DC motors due to ease of control and restrictions in field of application. By controlling the angular velocity of a motor at desired instant, we can bring about precise control over the mechanical system. The library is trying to implement this control for motor drivers that are frequently used in the club.

OBJECTIVES

The library has control variables for the ease of calibration for different systems. Control factors are set so that it can be easily modified to change the behavior of Ramping as well as Running of motor. The modular approach helps in easily use the library for different drivers like Sabertooth and L293d.

REQUIREMENT

Positioning the slider system within 1mm accuracy with a test load of about 2.5 kg as dead weight.

LOOKING AHEAD

The library could be used to control any motor driven system. For example the library could use to directly control an encoded differential drive. The application is limited only by imagination.

Overview

Ramping is used to smooth out the motion of the motor. The main advantage of using ramping is to prevent sudden changes in current which causes adverse effects on motor's life and to avoid jerks which is not good for the mechanical system.

SOFTWARE

Since the velocity of motor can be controlled using PWM, so using analogWrite() for L293D and L298 control and USBsabertooth library is used to control Sabertooth 2x32.The PWM corresponding to the desired velocity is given to the motor driver at the desired time till it changes from initial to final rpm.

Position control is achieved using ramping up to an optimal velocity and then ramping down. The code is compatible with Arduino Mega or above.

The complete position control library consists of two parts:

- **Ramping.h** – helps to change the speed of motor in a controlled manner. It contains definition for the functions by which the ramping is implemented, we have developed two types of ramping headers, using time and position as independent variable respectively.
- **Goto.h** – uses the ramping header and chooses the method by which the motor should move from one position to another.

HARDWARE

- **Igus Slider attached through a spring coupler**
- **Sabertooth2x32.**
- **Maxon motor** (Part number 236669) **with Planetary Gearhead** (Part number 166162)
- **Encoder HEDS 5540**
- **Arduino Mega.**

Theory

A motor driven slider, here mentioned as system, has a non-varying properties like power of motor inertia and friction which determine the limiting acceleration of the system. In order to smoothly control the system this limiting are found. The limiting acceleration when the motor is increasing its angular velocity and when it is decreasing its angular velocity are different. This is due to the fact that the frictional factor acts with and against the direction of acceleration during different times.

Power is given as pluses at high frequency to the motor, the ratio of time of application of power to the time period is determined by the PWM value given to the motor driver, which is here referred as PWM of motor. The angular speed of motor is linear function of PWM of the motor, so in order to control the system, the PWM is varied. So sudden discrete change in the value of PWM can assumed to produce jerks in motion. This is achieved by the ramping library, which gives continues change of PWM.

IMPLEMENTATION OF RAMPING

In order to control the acceleration different types of ramping can be used like linear ramping, S-ramping and so on. In linear ramping the velocity will be a linear w.r.t time. S-ramping has the feature that the velocity changes from initial to final through an S curve, if plotted against time. S-ramping helps in removing sudden change in velocity so that the jerks will be removed from the system. The function x-sin(x) has this properties so this function is a possible candidate. Another way to do this is by using different functions for different parts of S-curve and solving to find the parameters such that the resulting curve is smooth. But the problem with this is that it will require much more computing than a simple function, at this point it is clear that x-sin(x) gives a time advantage.

S-ramping

Once the limiting acceleration is found, if the motor is required to change angular velocity from say v1 to v2, the acceleration is smoothly increased to the limiting value and decreased back to zero. This is done so that the velocity changes from v1 to v2. The acceleration is changed just like 1-cos (k).Here k will be a function of time. See the fig. attached aside.

plot	$1 - \cos(x)$	$x = 0$ to $2\,\pi$

Plot:

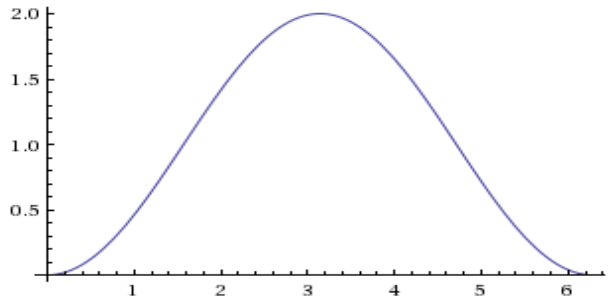


FIG.

THEORY

The corresponding change in velocity will be in the form of k-sin (k). (See the graph)

plot	$x - \sin(x)$

Plots:

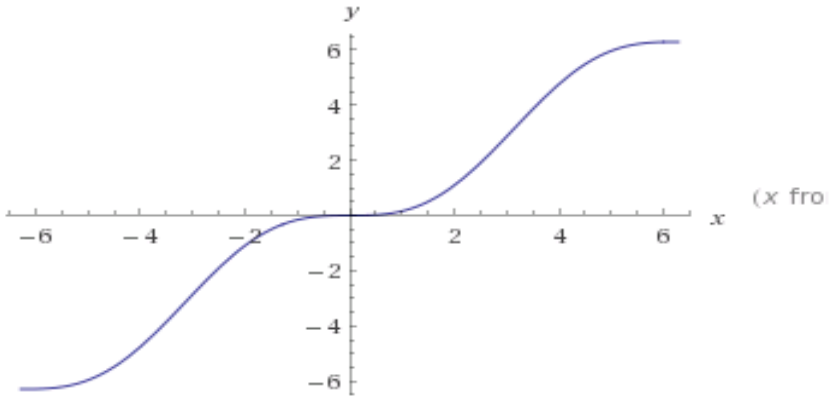


FIG.
We will be interested in the part from 0 to 2pi.

The library is to implement the required change in velocity so that it follows the required ramping path. The ramping can be done w.r.t time or w.r.t distance. If the ramping is to be controlled as a function of time and since we have the control on only PWM of motor, we will map the required change in angular velocity as change in PWM and the PWM values are altered so that it follows a S-curve with respect to time to achieve the desired change in angular velocity.

Similarly if the ramping is to be implemented with respect to distance, PWM corresponding to each position is given to the system. But one set back of the library was the in ability to implement S-ramping by directly solving the equation and eliminating time depending factor. This was due to the complexity of equation

Velocity, V= t-sin (t)
Integrating wrt 't'
Distance, D= $t^2/2$+cos (t) +C,

But here we can see that it is not possible to solve by eliminating t. The next option was to plot velocity and distance for different values of t. and find a best approximate function. In the development of library this process is used by the help of regression tools. A third degree polynomial is obtained, the coefficients depend upon the constant along with t in the equation. This equation is used to ramp the system w.r.t distance.

POSITIONING OF SYSTEM

For positioning of the system, the idea is to reach the required position with minimum time. So a maximum velocity is found the system is ramped up to it, then the speed is maintained till it reaches a position from

which it can safely ramp down to the required position. One problem with this is that there is a minimum PWM below which the system won't start to move, termed the minimal velocity. This is due to the stall torque due to the friction. So during implementation of the library, during ramp up the minimal velocity is given directly and then from that point it is ramped up till the possible maximum velocity. During ramp down it is ramped down till the minimal velocity and the system is stopped just after the required position is reached. This also helps in the motion of very small distances accurately, otherwise sometimes the velocity might come out to be less than the minimal velocity for small distances and the system may not move at all.

Library Structure

RAMPING LIBRARY

The library is made in the form of class. This header has all the functions to smoothly ramp the system using the desired ramping type (default mode is S-ramping). The function which is to be used for ramping is stored as a macro, this also helps in changing the function for ramping. Library can store two different functions at the same time, the header currently contains linear ramping as optional mode. During initialization the required function is selected. It has initializing functions and loop functions. Loop functions contain the ramping function which, if ramping is to be done, will find the required PWM and update the PWM variable. This PWM can be directly applied to any driver.

Methods of implementation

The arduino or platforms similar to that has a looping/interrupt method of execution rather than time division or priority based execution. So it was extremely important to implement the ramping so that it won't hinder any other valuable processing at the same time. There was two options to do this:

a) Using default Interrupts which are executed after fixed time
b) Using a function which is repeatedly executed in loop.

Interrupt Method

Arduino has internal timer interrupts which runs to maintain the clock within. The same clocks can be used to trigger functions at certain intervals of time. The merit is that the library ramps at specified intervals, as the interval time is minimized the ramping will become more fine. Since the encoder interrupt is also dealt with the same arduino this may cause loss of data from encoder. It may also lead to hanging of arduino as the interval is minimized and may interfere with communication since it also uses interrupt method for communication.

Loop Method

If the ramping is done as a function in loop, there will be no influence on other functions or interrupts running on Arduino. But number of ramping calls (Smoothness) will depend upon the time taken to execute the loop, which will decrease as the size of code increase in the loop. Considering the merit of Loop method, Loop method is implemented the library.

Initializing functions

a) Ramp::Ramp(int Function_Number, double lowerbound, double upperbound)

This constructor selects the function which is to be used for ramping. The lower bound and upper bound determines which part of the function, based on interval on x-axis, is to be used for ramping. Eg: when function is x-sin(x) lower bound is selected as 0 and upper bound as 2*pi. The library can be easily used for different drivers, for that during the initialization the range of PWM is to be specified. The output variable where the instantaneous PWM is updated is controlled using pointers so that different functions can update or use the same variable.

b) Ramp:: pass (double * Current_position, int *PWM_of_motor, int minimumspeed)

This function associates the global variables like current position of system from encoder, instantaneous PWM of motor and also the minimum speed which is discussed earlier. The library ramps down till minimum speed and leaves the system there.

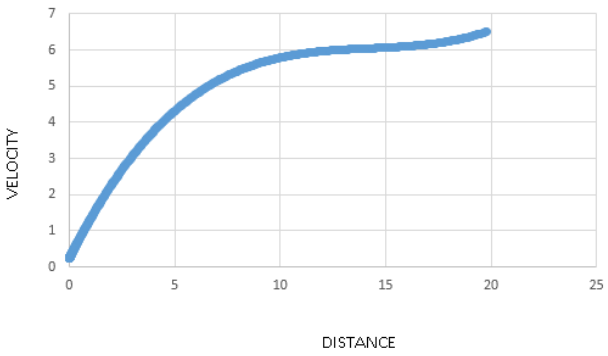
Loop Functions

a) Ramp:: SetSpeed(double final_position, int PWM_final)

This function is used to initiate ramping as well as set the final PWM at the final location. When the function is called it sets a flag to true, this flag switches the ramping to true. The current position, final position and the velocities are stored into appropriate variables. These variables are later used by the Ramp::rmp() function to give appropriate PWM at appropriate location. The SetSpeed() should be used such that the final position is in the direction of motion.

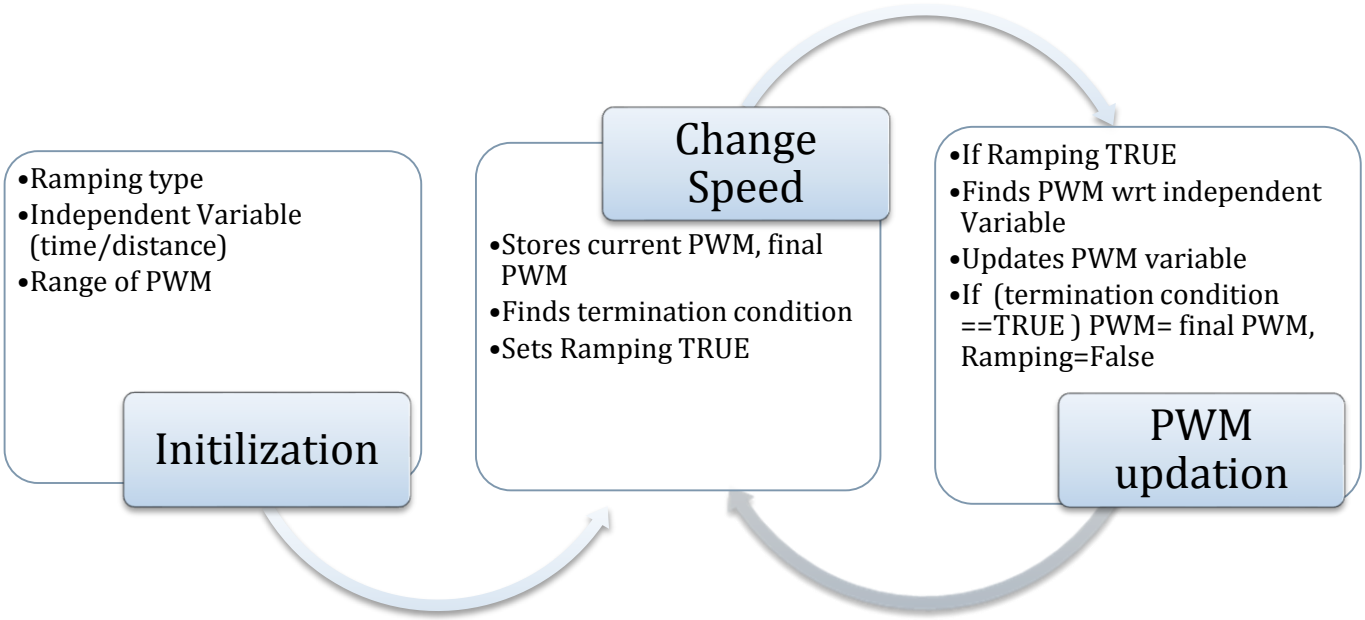
b) Ramp::rmp(boolean flag)

If ramping flag is set to true and a true value is passed as input to this function, the function will use the current position and finds the corresponding PWM using the initial PWM, final PWM, starting point and ending point. Consider the figure. The function maps the 0 - 20 t0 initial position –final position as well as 0 – 6.28 to initial PWM - final PWM. So by the mapping the PWM for required position is found. If the position is crossed the function writes PWM as final PWM. Also if at any point the PWM required is less than minimum PWM, PWM is written as minimum PWM.



LIBRARY STRUCTURE

SUDO CODE



POSITION CONTROL LIBRARY

This header uses real time variables to control the system with the help of predefined ramping library. The library is coded as a class, which has also both initialization function which is a constructor and loop function. Since the library uses ramping w.r.t distance limiting factors are to be found and replaced in appropriate place. This factors depend upon the system.

System Factors

a) **Distance Ramp up factor (AA)**
This factor is used to predict how much distance will be covered if the system changes the velocity to its maximum from rest without ramping. It has been experimentally found that for different PWM this distance increases linearly. So the distance vs PWM is plotted and the appropriate linear coefficient is found.

b) **Distance Ramp down factor (BB)**
Similar to the factor explained above; BB is the distance covered by the system due to inertia, after the PWM is set to 0 while the system was at moving at maximum PWM. This distance versus PWM is plotted and the linear coefficient is found.

c) **Minimum speed**
This is the minimum PWM at which the system can move. This is used for ramping as well as while traversing very small distances.

LIBRARY STRUCTURE

Initialization Function

RampPositioning ::RampPositioning(double*current_position, int*speed_at_instant, int PWM maximum, int PWM minimum, int minimumspeed)

This constructor links the global variables which has the instantaneous position as well as instantaneous speed of motor. These are made as pointers so that the same can be either used or modified by other functions. Pwm maximum and PWM minimum limits the spread of PWM. Minimum speed is to be updated at this point while other system parameters are to be edited in the function.

The reason for this is that distance factor might behave differently for different systems and we cannot always relay upon linear property between PWM and stopping/starting distances.

Looping Function

RampPositioning ::Startpositioning(boolean flag, double final position, int final PWM)

This function is to bring the bot to a particular position and leave it there at final PWM, for the project this PWM is set to zero in order to stop the system at that position. The function when called initially chooses the method by which it has to reach the final position. The reason for adopting this policy is that

a) Initial direction of PWM could be opposite to the required direction of motion
b) Initial velocity could be so high that it overshoots the final position

Identification of Method (Path)

Function checks whether direction of PWM is opposite to the direction of motion or the initial velocity is high that it overshoots final position, if yes system is stopped and path 1 is selected. (Path 1 is discussed below). If no, the maximum possible speed is calculated. The calculation is done by using inverse mapping with the help of distance ramp factors. If this maximum speed is less than minimum speed of the system, path 3 is selected else path 1 is selected.

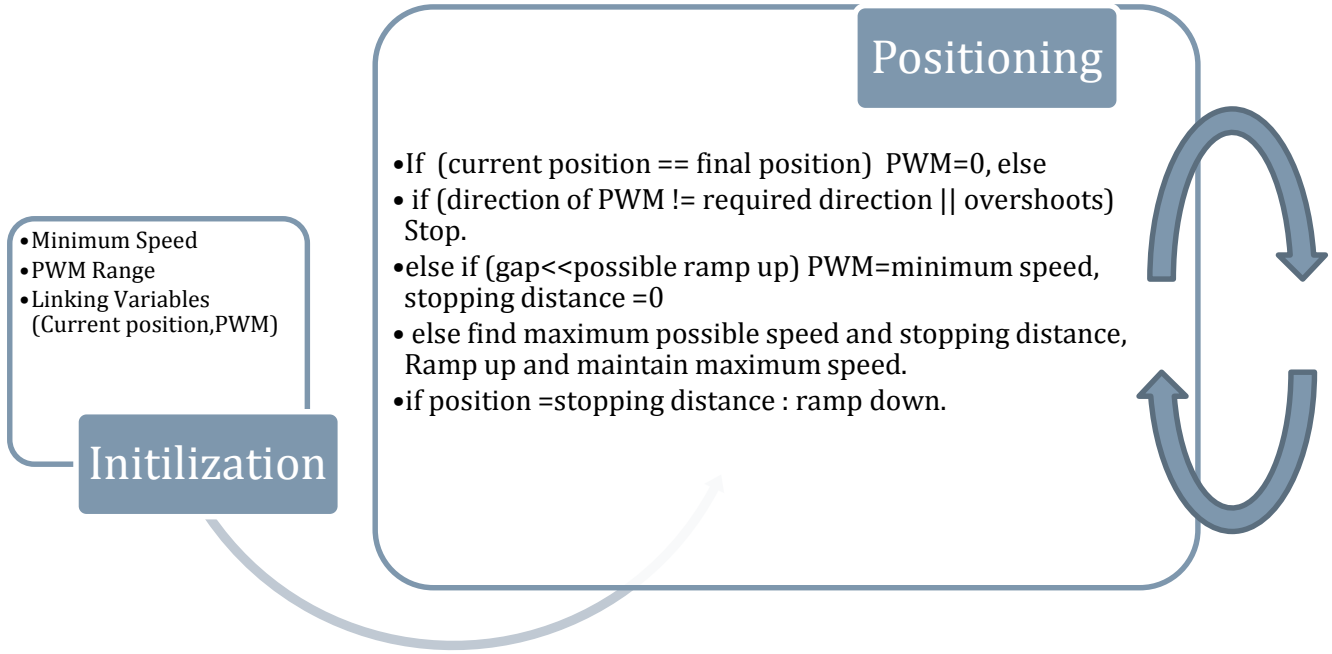
Paths

Path 1: System is ramped up till the maximum speed and moves in the direction of final position. When the system reaches the ramp down point, (position is found using distance ramp down factor) it is ramped down till the final position and stopped.

Path 3: System is maintained in the minimum speed till it reaches the final position. When it reaches the final position the motor is stopped.

LIBRARY STRUCTURE

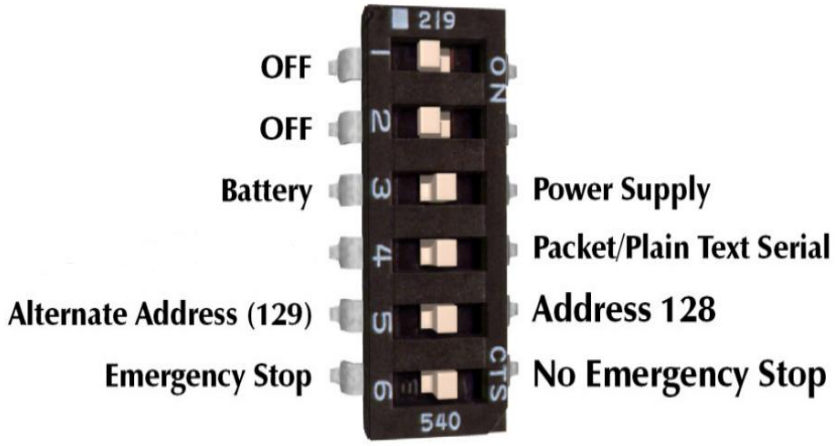
Sudo Code



Hardware and Circuit

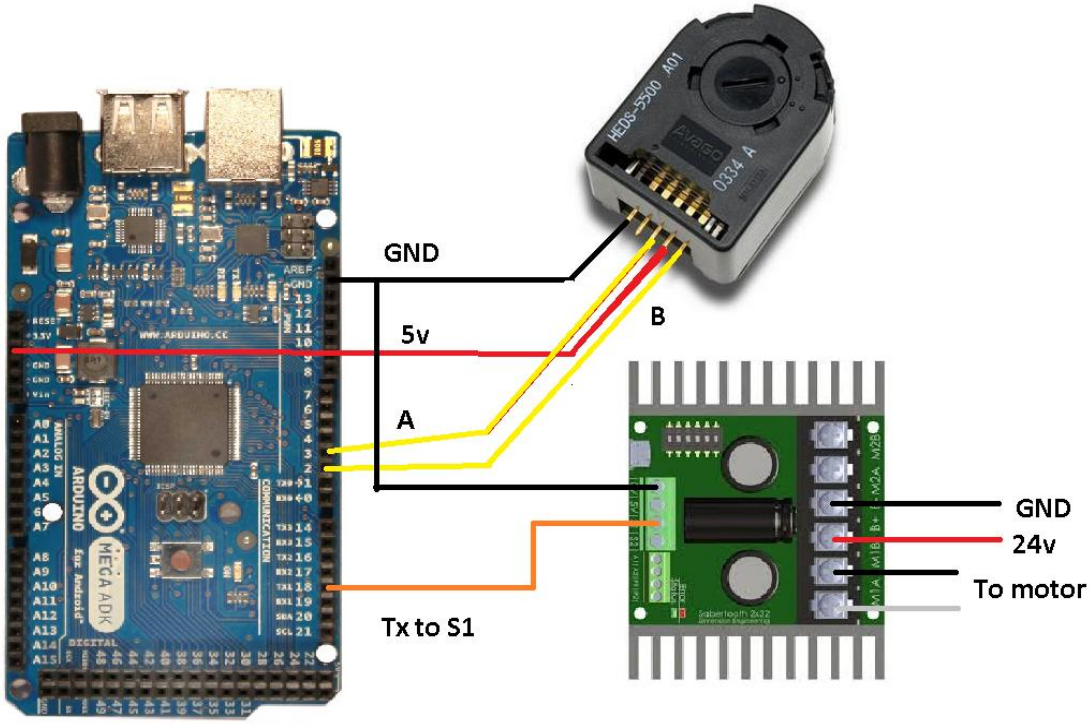
HARDWARE SPECIFICATIONS:

Hardware	Specification
Maxon Motor	212 mNm stall torque, 6400 rpm, 24V
Planetary Gearhead	16:1 ratio
Encoder HEDS 5540	500 ppr
Sabertooth 2x32	Packet Text Communication



1 Dip Pin Configuration:

HARDWARE AND CIRCUIT

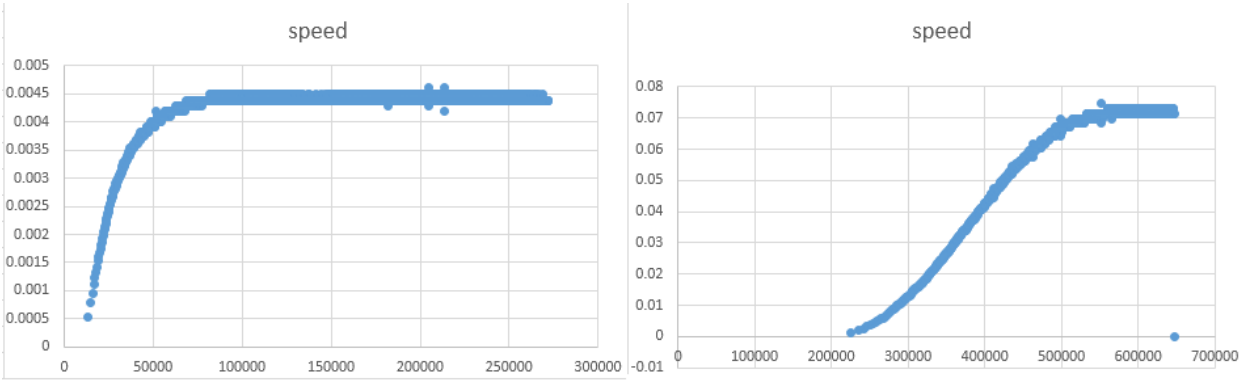


2 Circuit Diagram

Results

RAMPING

After the application of ramping the angular speed vs time of maxon motor was plotted using the help of encoder. The results are shown below



X axis is time in microseconds while y axis is the angular velocity. In the right hand side graph (one in which S-ramping is implemented) the velocity of the system is increasing smoothly preventing any jerks acting on system.

POSITION CONTROL

The accuracy of position control depends upon the system on which it is implemented, during the project it was implemented on igus slider driven by maxon motor and Sabertooth 2x32. The following results are obtained on them. We were able to stop the slider within an accuracy of 0.2 mm with a weight of 2Kg at maximum speed of the motor.

As far as the library is concerned, the library can be used for any type of driver like Sabertooth or L293d. Also it is easy to calibrate the library for any system.

Future Possibilities

- **Implementation using timed interrupt**
- **Control of Differential drives**
- **Use of PID with fine tuning**

References and Links

References:

- Use of Multiple Functions for Ramping adapted from Summer project by Varan Gupta.
- Trajectories for ramping- PDF by Vaibhav Gupta.
- https://learn.sparkfun.com/tutorials/pulse-width-modulation
- http://www.maxonmotor.in/maxon/view/content/service-academy-motor

Links:

- Ramping with respect time Code
- Position Control Code
- Motor Data Sheet
- Other Related Files

CONTACT INFORMATION

Contact Information

AMAL GEORGE M	ADITYA JAIN
	
Tel [9891050229] [amalgeorgem94@gmail.com]	**Tel** [7503292123] [adiclaws007@gmail.com]

Club Details

Robotics Club
IIT Delhi
Hauz Khas
http://roboticsclub.iitd.ac.in/