An Investigation Of The Implementation Of S-Curve Motion Profiling Model In Robotics Applications

Research Question: How can the implementation of the S-curve motion profiling model be applied with active target displacements in autonomous robotic chassis?

Subject Area: Physics

Word Count: 3975

Table of Contents

Introduction **3**

Background **4**

How objects stop and accelerate 4

S-Curve Motion 5

SECTION I - Mathematical Modeling **6**

Defining Functions 6

Manipulating Parameters 11

SECTION II - Virtual Simulation **12**

Pseudocode for general algorithm logic See Appendix for full program in Python 12

Table 2 - Output Velocity Data Table 14

Graph of Expected velocity vs. output velocity 15

SECTION III - Physical Experimentation **16**

Materials 17

Procedure 17

Data Collection 18

Table 3 - Raw Data Table 19

Graphs of horizontal displacement(m) plotted against time(s) plotted using Table 2 19

Discussion 21

Data Interpretation 21

Evaluation 21

Future Improvements 23

Real-Life Application 23

Conclusion **24**

References **25**

Appendix **27**

Introduction

Motion profiling plays a vital role in the field of robotics, automation, and control systems, enabling precise and controlled movement of objects or machines. Point-to-point movement is an essential part of mechatronics and robotics applications(Blejan and Blejan, 2020). By inputting the target displacement into the embedded system, the vehicle should be able to travel the desired distance at a set speed. Among the various motion profiles, the S-curve motion profile stands out for its smooth acceleration and deceleration characteristics, offering improved performance and reduced stress on mechanical systems. Simulating and analyzing S-curve motion profiling is essential to ensure accurate trajectory planning and execution.

Last year for a robotics competition, I had a lot of trouble programming the robot because it was experiencing tremendous amounts of momentum whenever the turret spun and the arm extended to grab objects. Towards the end of the season, we were made aware of the s-curve model but ran out of time to implement it on our robot. Using this opportunity, I wanted to experiment with this model on a robot drivetrain, as it was the most familiar thing to me. Knowing the need to promote robotics outside the lab environment, I started to investigate the areas of applications for S-Curve motion profiling and found that there is a high need for “soft” grasping for delicate objects on production lines. I was quite inspired by this topic as it did seem like a more relevant and tangible topic for everyday applications. This lead me to the research question for this extended essay: **how can the implementation of the S-curve motion profiling model be applied with active target displacements using an autonomous robotic chassis?**

Background

How objects stop and accelerate

Objects accelerate and decelerate due to the action of forces acting upon them. Acceleration refers to a change in an object's velocity over time, and deceleration is a specific type of acceleration where the object's velocity decreases.

When an external force is applied to an object, such as a push or a pull, it can cause the object to accelerate. According to Newton's second law of motion, the acceleration of an object is directly proportional to the net force acting on it and inversely proportional to its mass. Mathematically, this relationship is expressed as: F = ma where F is the net force acting on the object, m is the mass of the object, and a is the acceleration produced. The force applied to an object determines how much it will accelerate. If a greater force is applied, the object will experience a larger acceleration, and if a smaller force is applied, the acceleration will be smaller.

Newton's third law posits that for every action, there is an equal and opposite reaction. In the dynamics of acceleration and deceleration, which signify changes in velocity, this law holds sway. Acceleration, indicating an uptick in velocity, is instigated by an action force, met with an equal and opposite reaction force per Newton's Third law from the frictional force from the road. Notably, the relative directions of velocity and acceleration hold significance(Hamm). In cases where both vectors align, the object speeds up. Conversely, if they oppose each other, deceleration occurs. This phenomenon is evident when applying brakes to a moving car: the action force of friction acts opposite to the velocity, leading to deceleration (Hamm). Newton's third law ensures that these forces occur in reciprocal pairs, shedding light on the intricate interplay of forces and their role in altering an object's motion.

As objects move or gain displacement, they obtain a velocity, which is the change in its position relative to time; with velocity, comes acceleration, which is the rate of change in velocity relative to time. The general equation to find the relationship between the three is displaced in Table 1.

Table 1. General equations for the conversions between displacement, velocity, and acceleration.

	Differentiate	Integrate
Displacement	$x(t)$	$x(t) = x_0 + \int_0^t v\,dt$
Velocity	$v(t) = \frac{dx}{dt}$	$v(t) = v_0 + \int_0^t a\,dt$
Acceleration	$a(t) = \frac{dv}{dt} = \frac{d^2x}{dt^2}$	$a(t)$

S-Curve Motion

S-curve equations are a third-degree polynomial mathematical model used to optimize the acceleration, deceleration, and turning movements in robotics and automation. The s-curve equation can take target displacement as one of its inputs, hence the model can plan and optimize the trajectory for automated vehicles by predicting the appropriate speed, acceleration, and deceleration for each segment of the path while reaching its desired displacement. Therefore, decreasing the residual vibration caused by internal inertia.

In current applications, the s-curve equation is often used to model the vehicle's response to different driving scenarios, such as sudden stops, lane changes, and obstacles in the road; rather than abruptly coming to a stop, the function can gradually slow the object down. By optimizing their movements, this model can help to reduce wear and tear on the vehicle, conserve energy, and improve passenger comfort, and especially beneficial in transporting cargo.

Looking into the near future, if all vehicles can utilize the s-curve equation, building a network of their relative positions to each other, one vehicle can predict and respond to the movements of other vehicles. The ability to accurately model and predict the movements of autonomous vehicles and other vehicles on the road is critical for ensuring the safety and reliability of autonomous driving systems.

Additionally, the s-curve equation plays a significant role for robotics engineers, as it can help to improve the performance, reliability, and safety of robotic systems. Robotic arms have been populated in factories worldwide, and even more in aerospace operations. Thus, controlling the movement of a robot arm, allowing it to move smoothly and accurately from one point to another with the help of inverse kinematics becomes a critical point engineers are trying to break. With the help of S-Curve models, Robots can then perform complex tasks with greater accuracy and efficiency, such as assembling small components, sorting and packaging products, and performing surgical procedures. S-curve motion can also be used to optimize the movement of mobile robots, such as underwater autonomous vehicles (UAVs) and automated guided vehicles (AGVs), allowing them to navigate complex environments and avoid obstacles while moving efficiently.

I - Mathematical Modeling

Defining Functions

Traditionally, static acceleration is used in a closed-loop system, where the instantaneous velocity of an object reaches its target velocity in a linear manner. This approach is efficient and simple, and outputs satisfactory results when implemented in real life. However, as a current challenge in motion control, vibration and over-carried momentum results in an overshooting of velocity, and often positions (Doang, 2016). Theoretically, the trapezoidal velocity profile is the most time-efficient – reaching the desired displacement in the shortest time. That said, when considering the jerk of the profile, as illustrated in Figure 1, when time is at t_0, t_1, t_2, and t_3, the jerk constantly advances from zero to infinity in one instant; this is unrealistic. This sudden jump in the jerk constant will result in the velocity overshooting at times t_1, and t_3, and by extension, whenever velocity changes direction. Therefore, the manipulator's residual vibrations, caused by internal inertia, will delay the time for the object to finally adjust to its desired displacement. (Veciana & Cardona, 2012)

The vibrational energy and velocity overshoots can be decreased by using a third-order polynomial S-curve equation(Fig. 1b). In this way, the change in acceleration becomes linear, resulting in a first-order jerk equation, and a second-order velocity profile.

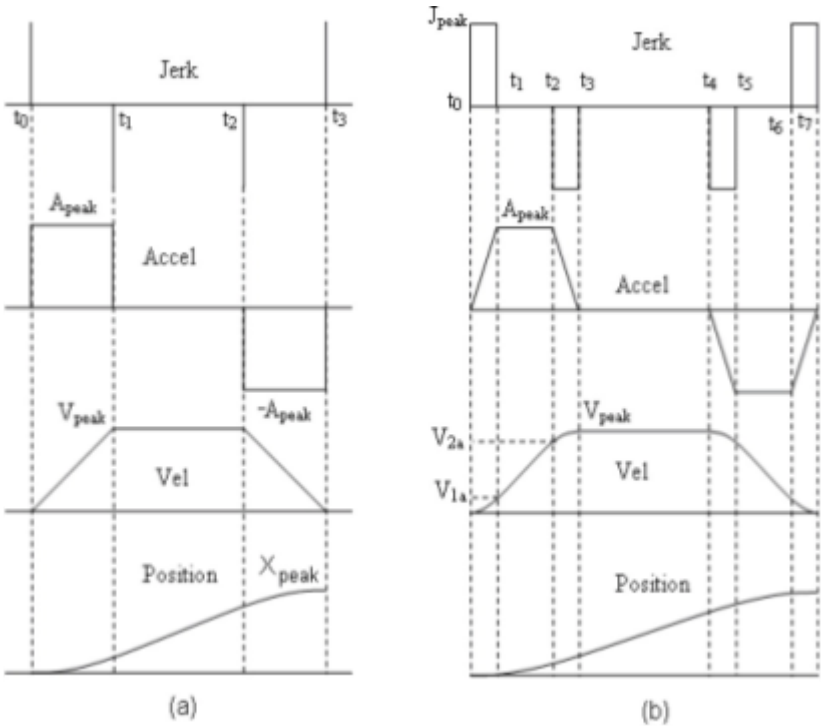


Figure 1. Trapezoidal Velocity Profile (a) and cubic S-Curve Profile (b)
From (Doang, 2016)

By optimizing the time it takes to reach a certain displacement using the max velocity and acceleration, several equations may be formed that guide the motion. As the S-curve profile can be represented using a piecewise system, the following equation can be derived from the S-curve acceleration piece-wise equations. In order to create these equations, one may simplify the prompt through the basic geometry of the trapezoids in the acceleration profile.

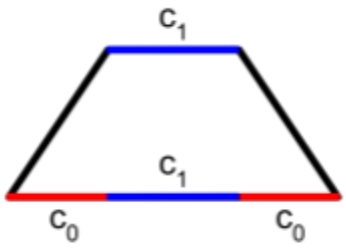


Figure 2. Geometric representation of a portion of the acceleration profile

Let c_2 be the start of the next trapezoid reflected over the horizontal axis; let j be the jerk constant in the profile which is the derivative of the acceleration profile, thus adjusting the slope of the acceleration profile. Plotting the seven segments into a piecewise function for acceleration, one can find the following function.

$$A(t) = \begin{cases} jt & 0 \le t < c_0 \\ jc_0 & c_0 \le t < c_0 + c_1 \\ -j(t - 2c_0 - c_1) & c_0 + c_1 \le t < 2c_0 + c_1 \\ 0 & 2c_0 + c_1 \le t < c_2 \\ -j(t - c_2) & c_2 \le t < c_0 + c_2 \\ -jc_0 & c_0 + c_2 \le t < c_0 + c_1 + c_2 \\ j(t - 2c_0 - c_1 - c_2) & c_0 + c_1 + c_2 \le t \le 2c_0 + c_1 + c_2 \end{cases}$$

Using these variables, this piecewise function accurately depicts the acceleration profile. Thus, by integrating this equation relative to time, the velocity profile is created. In addition, one may notice that $a_{max} = jc_0$. By finding the definite integral of each of the seven segments of the acceleration profile, one may recognize that $v_{max} = a \cdot (c_0 + c_1) = j \cdot c_0 \cdot (c_0 + c_1)$ and thus when integrating once more for the displacement profile, $x = v_{max} \cdot c_2 = j \cdot c_0 \cdot (c_0 + c_1) \cdot c_2$. Rearranging these equations to isolate for c_0, c_1, and c_2, one can define the seven segments to the profile in terms of a_{max}, j, v_{max} and d.

$$c_0 = \frac{a}{j}, c_1 = \frac{v}{a} - \frac{a}{j}, \text{ and } c_2 = \frac{d}{v}$$

Thus, but substituting the new equations for c_0, c_1, and c_2, one can define the S-Curve equation using relative parameters. Finally, with reference to Table 1, one needs to calculate the definite integral representing velocity from the acceleration profile using the new upper and lower limits. Replicate the

process once more to find the displacement using the velocity piecewise function. The final piecewise functions for displacement, velocity, and acceleration with the new segment parameters are as follows (Beljan, 2020).

$$A(t) = \begin{cases} jt & 0 \le t < \frac{a}{j} \\ a & \frac{a}{j} \le t < \frac{v}{a} \\ -j(t - \frac{v}{a} - \frac{a}{j}) & \frac{v}{a} \le t < \frac{v}{a} + \frac{a}{j} \\ 0 & \frac{v}{a} + \frac{a}{j} \le t < \frac{d}{v} \\ -j(t - \frac{d}{v}) & \frac{d}{v} \le t < \frac{d}{v} + \frac{a}{j} \\ -a & \frac{d}{v} + \frac{a}{j} \le t < \frac{d}{v} + \frac{v}{a} \\ j(t - \frac{d}{v} - \frac{v}{a} - \frac{a}{j}) & \frac{d}{v} + \frac{v}{a} \le t \le \frac{d}{v} + \frac{v}{a} + \frac{a}{j} \end{cases}$$

$$V(t) = \begin{cases} \frac{j}{2}t^2 & 0 \le t < \frac{a}{j} \\ a(t - \frac{a}{2j}) & \frac{a}{j} \le t < \frac{v}{a} \\ -\frac{j}{2}(t - \frac{v}{a} - \frac{a}{j})^2 + v & \frac{v}{a} \le t < \frac{v}{a} + \frac{a}{j} \\ v & \frac{v}{a} + \frac{a}{j} \le t < \frac{d}{v} \\ -\frac{j}{2}(t - \frac{d}{v})^2 + v & \frac{d}{v} \le t < \frac{d}{v} + \frac{a}{j} \\ -a(t - \frac{d}{v} - \frac{a}{2j}) + v & \frac{d}{v} + \frac{a}{j} \le t < \frac{d}{v} + \frac{v}{a} \\ \frac{j}{2}(t - \frac{d}{v} - \frac{v}{a} - \frac{a}{j})^2 & \frac{d}{v} + \frac{v}{a} \le t \le \frac{d}{v} + \frac{v}{a} + \frac{a}{j} \end{cases}$$

$$D(t) = \begin{cases} \frac{j}{6}t^3 & 0 \le t < \frac{a}{j} \\ \frac{a}{2}(t - \frac{a}{2j})^2 + \frac{a^3}{24j^2} & \frac{a}{j} \le t < \frac{v}{a} \\ -\frac{j}{6}(t - \frac{v}{a} - \frac{a}{j})^3 + v(t - \frac{v}{2a} - \frac{a}{2j}) & \frac{v}{a} \le t < \frac{v}{a} + \frac{a}{j} \\ v(t - \frac{v}{2a} - \frac{a}{2j}) & \frac{v}{a} + \frac{a}{j} \le t < \frac{d}{v} \\ -\frac{j}{6}(t - \frac{d}{v})^3 + v(t - \frac{v}{2a} - \frac{a}{2j}) & \frac{d}{v} \le t < \frac{d}{v} + \frac{a}{j} \\ -\frac{a}{2}(t - \frac{d}{v} - \frac{a}{2j})^2 + v(t - \frac{v}{2a} - \frac{a}{2j}) - \frac{a^3}{24j^2} & \frac{d}{v} + \frac{a}{j} \le t < \frac{d}{v} + \frac{v}{a} \\ \frac{j}{6}(t - \frac{d}{v} - \frac{v}{a} - \frac{a}{j})^3 + d & \frac{d}{v} + \frac{v}{a} \le t \le \frac{d}{v} + \frac{v}{a} + \frac{a}{j} \end{cases}$$

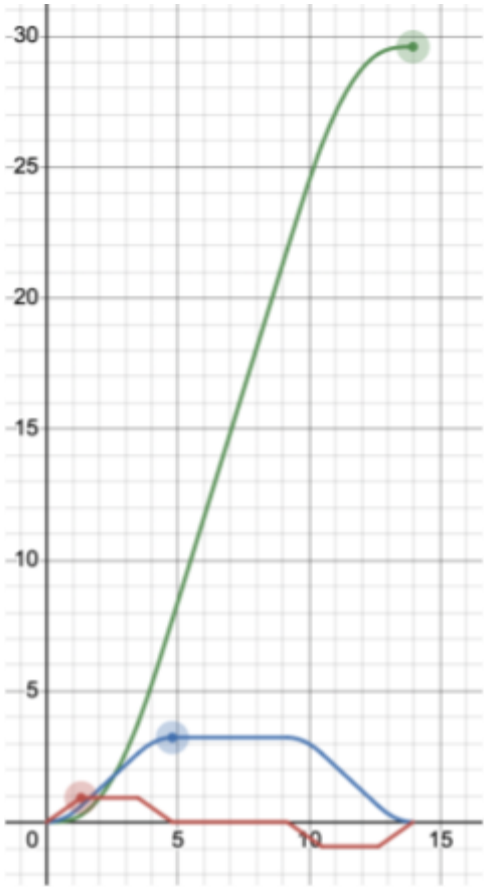


Figure. 3 Graph of S-Curve Equations of Displacement,
Velocity, and Acceleration with respect to Time in DESMOS

However, the motion profile is still not complete as there are bounds that need to be integrated within the system that manages the v_{max} and a_{max} based on the displacement and jerk given. Since these piecewise functions are assumed to reach their max velocity and acceleration in order to minimize time, they will break when the displacement is short because there will not be enough distance to reach these maxima. Thus, bounds are required to reduce these maxima and to implement them, one may imagine three different systems for scaling displacement.

Manipulating Parameters

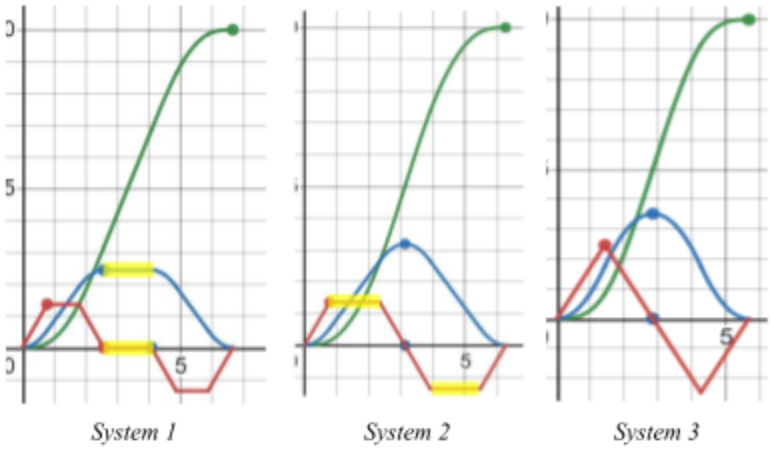


Figure 4. 3 Systems of S-Curve equations without bounds for parameters

In the first system, the parameters v_{max} and a_{max} have no heavy restrictions and no bounds are needed. However, in the second system when $2c_0 + c_1 = c_2$ the v_{max} becomes heavily restricted, assuming that $v_{max} \geq \frac{a^2}{j}$. Solving for v_{max} and taking the positive root through the quadratic formula, the restrictions for v_{max} can be found as such: $v_{max} \leq \frac{-a^2+\sqrt{a^4+4adj^2}}{2j}$ (Mu, 2009). One may also realize that in the second system $c_0 + c_1 \geq c_0$. Thus, a new restriction can be found using the formula: $v_{max} \geq \frac{a^2}{j}$, though it is not necessary when scaling for displacement. In the third system, to scale the optimal system down, it is required to create bounds for a_{max}. By using the previous equation: $a^2 \leq v \cdot j$ and substituting the v_{max} upper bound and solving for a_{max}, the restriction on a_{max} can be expressed as $a_{max} \leq \sqrt[3]{\frac{dj^2}{2}}$. Finally, these are the bounds to v_{max} and a_{max}: $a_f = min(a_{max}, \sqrt[3]{\frac{dj^2}{2}})$ and $v_f = min(v_{max}, \frac{-a^2+\sqrt{a^4+4adj^2}}{2j})$ (Mu, 2009).

II - Virtual Simulation

One can simulate how an object accelerates and decelerates and its related motion profile using programming. One would use multiple float variables to store the necessary parameters, such as initial position, final displacement, maximum velocity, maximum acceleration, and maximum jerk. Additionally, the total duration or time interval for the motion will need to be specified. By using mathematical equations derived in section 1, one can create an algorithm which calculates the trajectory of the S-curve motion profile. For this essay, the quintic polynomial approach will be used to calculate the motion profile. However alternatively, one can use numerical integration techniques to approximate the trajectory based on the given constraints. By dividing the motion's duration into seven time intervals or phases, one can calculate the position, velocity, acceleration and jerk at a specific time. To visualize the motion profile, Smoothie.charts may be used to represent the S-curve motion in the simulation graphically. One would plot the position, velocity, acceleration, or jerk over time.

To implement the feedback loop in real-life scenarios, it is best to test its feasibility virtually before physical implementation. By creating a web-based simulation that allows for a dynamic input change, the program can be manipulated and predict the needed velocity to move towards a specific displacement. As the input will be dynamic, loops with functions are needed to ensure that the velocity and current displacement are updated periodically.

Pseudocode for general algorithm logic *See Appendix for full program in Python*

```
Program starts
Initialize variables, set desired jerk, maximum velocity and acceleration
Start loop, repeated every 1000 milisecond{
        Start timer t
        Retrieve target displacement value from slider
        Run S-Curve function{
```

```
                Determine phase by comparing the distance to the expected distance for each phase

                From phase, calculate output velocity

        }

        Find the change in displacement

        Update displacement

        Print calculated velocity

        Update time

}
```

Table 2 - Output Velocity Data Table

For maximum velocity = 3.7, maximum acceleration = 0.8, duration = 16, target displacement = 36.2

Time (s)	Output Velocity (m/s)	Output Displacement (m)
1	0.000000000	0.00000000
2	0.350103769	0.35010377
3	1.142992638	1.49309641
4	1.942992638	3.43608904
5	2.742992638	6.17908168
6	3.493759295	9.67284098
7	3.700000000	13.3728410
8	3.700000000	17.0728410
9	3.700000000	20.7728410
10	3.700000000	24.4728410
11	3.683632841	28.1564738
12	3.184034389	31.3405082
13	2.384034389	33.7245426
14	1.584034389	35.3085770
15	0.784034389	36.0926114
16	0.106408475	36.1990199

Graph of Expected velocity vs. output velocity

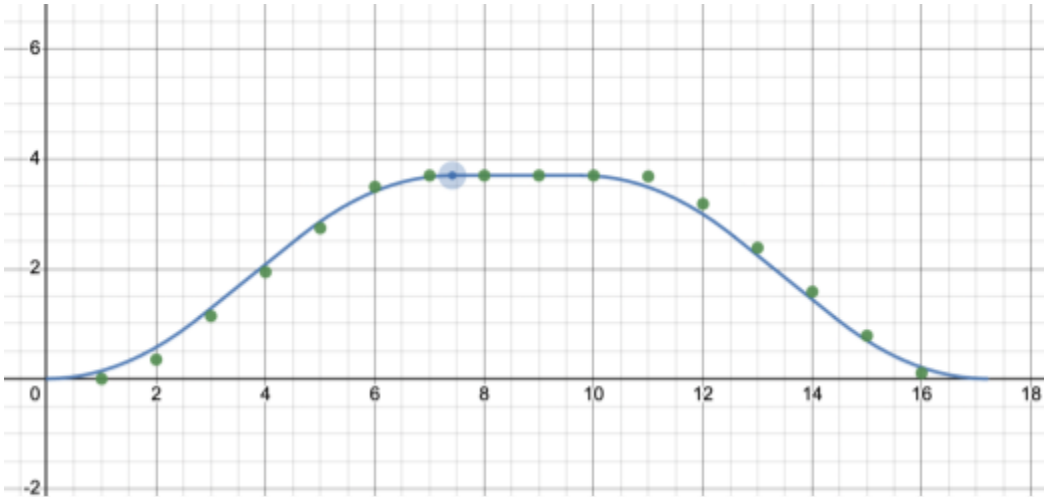


Figure 5. Program Simulated Value of Velocities (m/s) over 16s (Table 1) compared with Desired Change in Mathematical Value

Overall, the data points outputted by the algorithm are closely correlated to the velocity curve calculated. this demonstrates the success and efficiency and my program logic, as well as conciseness. thus, using the same logic, one could implement this algorithm into real-life situations, similar to the one mentioned in the ladder section of the essay. In the graph, the output velocity is similar to the one presented in DESMOS, where the data points present an overall trend similar to a second-degree polynomic piecewise function. However, there is a noticeable phase shift in the horizontal axis, representing time. what must consider the lack of precision presented by rounding errors within the program, and as the error accumulates, one must expect differences in the two outputs.

Consequently, the displacement graph will also reflect the inaccuracy, even greater in magnitude. The points plotted experience horizontal compression in comparison to the displacement curve. For instance, at t=16, the displacement reaches the desired value. However, in the expected displacement curve, the displacement does not reach that value until one second later. Thus, the displacement of the object is overshooting the calculated curve. However, this overshooting will likely be corrected in real-life applications as friction will cause the displacement to be lower than the velocity output.

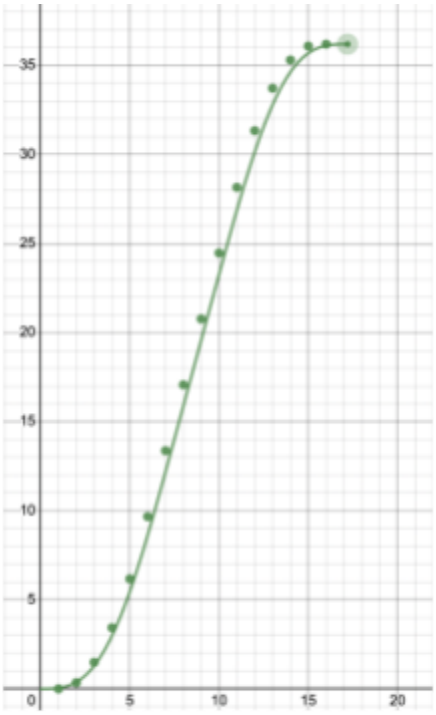


Figure 6. Computer Simulated Output Distance (m) over 16s (Table 1) compared to Expected Displacement from Mathematical Model

III - Physical Experimentation

The experiment is designed to evaluate the efficiency of the algorithm using optical sensors and DcMotors as a modular simulation of a full vehicle. The VEX motors were used to output the calculated velocity during the experiment, while VEX optical sensors were used to determine the initial target displacement. The experiment data will be collected via the encoder value, translated into displacement, the motor outputs per second. This value will then be compared to the simulated results to evaluate the efficiency of this real-life implementation. The experiment will be set up where the robot is programmed using the simulated program logic in C++, with a change in velocity every second with respect to the S-Curve velocity profile. Through analyzing the video footage recorded using the Tracker software, one can compute a relatively accurate velocity profile given a reference of displacement in the experimentation – measuring tape.

Materials

1	VEX Smart Motor & Gear Cartridges
1	VEX V5 Robot Brain
1	VEX pre-built chassis, 4 wheel omni-drive, 4 motors
1	VEX V5 Ultrasonic Sensor
1	10 x 10 x 3 block
1	Computer installed with VEXcode Pro V5
1	Measuring tape
1	Timer
1	Mico-USB to type-C cable

Procedure

1. The camera was fixed above the testing area in aerial view, capturing the whole testing field
2. The robot drivetrain is turned on with left motors initialized in ports 19 and 20, and right motors in ports 1 and 2, the gyro is initialized with port 16
3. Tape a blue sticker onto the robot brain for tracking purposes in analysis
4. A measuring tape is placed along the track for displacement references
5. Set the following parameters in the program for the sCurve(v, a, t, d, tj) function respectively:
    a. vel = 2.38
    b. acc = 0.8
    c. time = 16
    d. t_displacement = 15.1
    e. tech jerk = 35
6. The program was downloaded from VEXcode Pro V5 to the V5 controller brain using a micro-USB to type-C cable
7. Start video recording

8. Start the program on the robot controller by clicking "▶"
9. The robot controller should print expected velocity values on its screen, which may be utilized for troubleshooting
10. The video recording is stopped once the robot completes its trajectory
11. Steps 6-9 are repeated two more times
12. Video footage is exported to the computer for video analysis
13. The Tracker software is used to track the robot's position relative to time, thus propagating its velocity and acceleration
14. Data is exported into Table 3

Data Collection

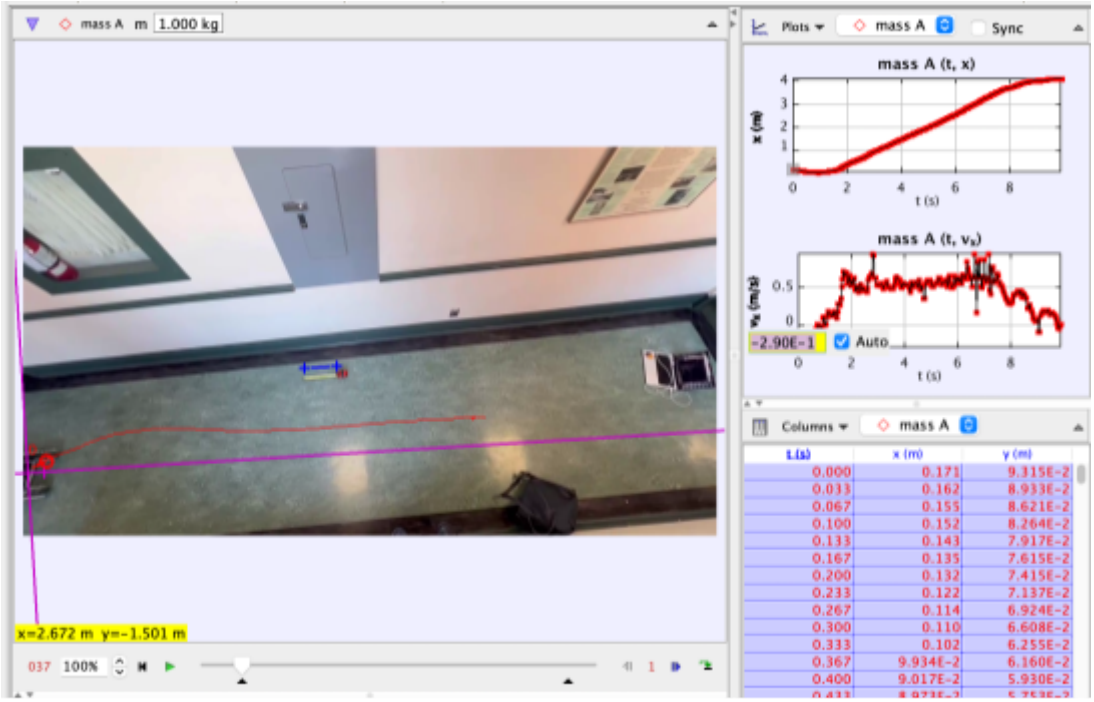


Figure 7. Sample of Tracker video analysis (Trial 1)

Table 3 - Raw Data Table

	trial 1			trial 2			trial 3	
t	d	vx	t	d	vx	t	d	vx
0.00E+00	1.71E-01	□	0.00E+00	2.76E-03	5.48E-01	1.58E+00	4.32E-01	-1.47E-01
3.33E-02	1.62E-01	-2.46E-01	3.33E-02	7.65E-02	5.48E-01	1.61E+00	4.32E-01	-7.82E-02
6.67E-02	1.55E-01	-1.53E-01	6.67E-02	7.62E-02	5.49E-01	1.65E+00	4.22E-01	1.33E-01
1.00E-01	1.52E-01	-1.71E-01	1.00E-01	7.59E-02	5.51E-01	1.68E+00	4.27E-01	-2.85E-02
1.33E-01	1.43E-01	-2.48E-01	1.33E-01	7.52E-02	5.52E-01	1.71E+00	4.31E-01	-2.43E-01
1.67E-01	1.35E-01	-1.67E-01	1.67E-01	7.37E-02	5.54E-01	1.75E+00	4.25E-01	-9.52E-02
2.00E-01	1.32E-01	-1.94E-01	2.00E-01	7.18E-02	5.57E-01	1.78E+00	4.15E-01	7.61E-02
2.33E-01	1.22E-01	-2.78E-01	2.33E-01	6.89E-02	5.57E-01	1.82E+00	4.18E-01	-5.08E-02
2.67E-01	1.14E-01	-1.92E-01	2.67E-01	6.28E-02	5.58E-01	1.85E+00	4.20E-01	-7.80E-02
3.00E-01	1.10E-01	-1.72E-01	3.00E-01	5.40E-02	5.62E-01	1.88E+00	4.15E-01	2.56E-01
3.33E-01	1.02E-01	-1.55E-01	3.33E-01	4.65E-02	5.63E-01	1.92E+00	4.15E-01	2.46E-01
3.67E-01	9.93E-02	-1.79E-01	3.67E-01	4.34E-02	5.67E-01	1.95E+00	4.32E-01	-8.97E-03
4.00E-01	9.02E-02	-1.44E-01	4.00E-01	3.95E-02	5.71E-01	1.98E+00	4.31E-01	-5.87E-03
4.33E-01	8.97E-02	-1.46E-01	4.33E-01	4.04E-02	5.72E-01	2.02E+00	4.32E-01	-1.84E-02
4.67E-01	8.04E-02	-2.90E-01	4.67E-01	4.15E-02	5.72E-01	2.05E+00	4.31E-01	-2.73E-02
5.00E-01	7.04E-02	-2.11E-01	5.00E-01	4.33E-02	5.77E-01	2.08E+00	4.30E-01	-3.69E-02
5.33E-01	6.63E-02	-2.13E-01	5.33E-01	4.34E-02	5.77E-01	2.12E+00	4.29E-01	-3.00E-02
5.67E-01	5.62E-02	-2.19E-01	5.67E-01	4.34E-02	5.79E-01	2.15E+00	4.28E-01	-2.54E-02
6.00E-01	5.17E-02	-8.36E-02	6.00E-01	3.99E-02	5.81E-01	2.19E+00	4.27E-01	-2.99E-02
6.33E-01	5.06E-02	-6.25E-02	6.33E-01	3.63E-02	5.82E-01	2.22E+00	4.26E-01	-5.16E-02

By exporting the data points from Tracker into the Raw Data Table (Table 3), one can plot graphs of the robot's position (d) relative to time (t).

Graphs of horizontal displacement(m) plotted against time(s) plotted using Table 2

Figure 8a

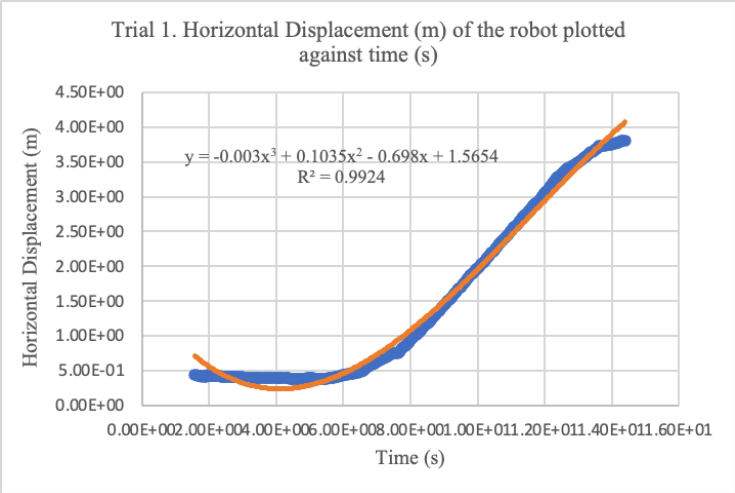


Figure 8b

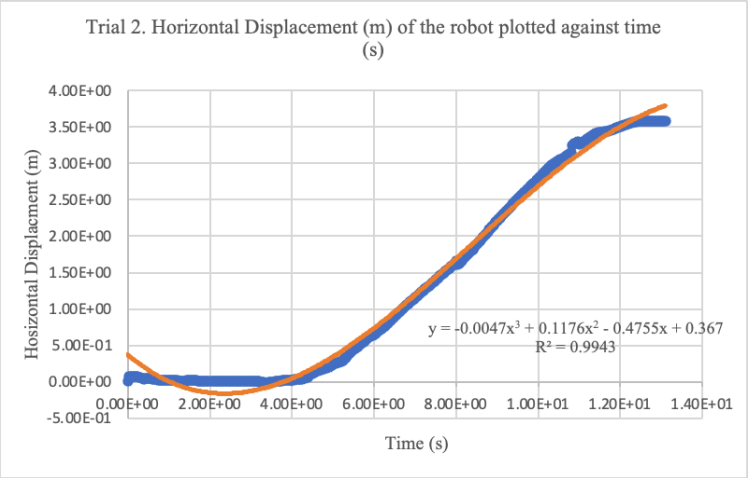


Figure 8c

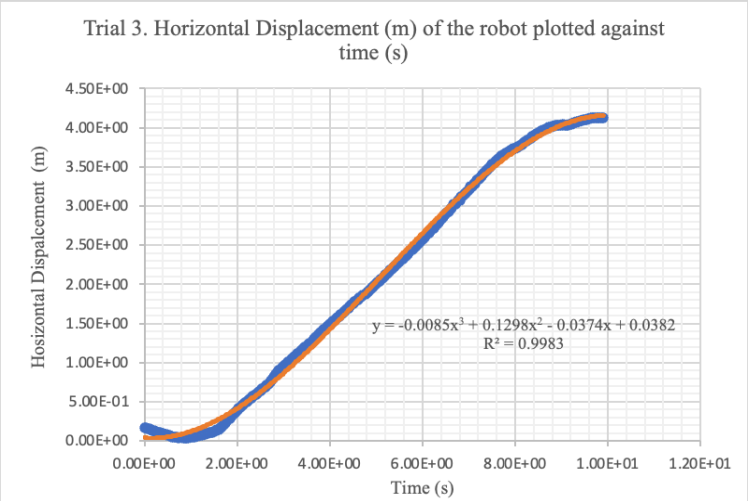
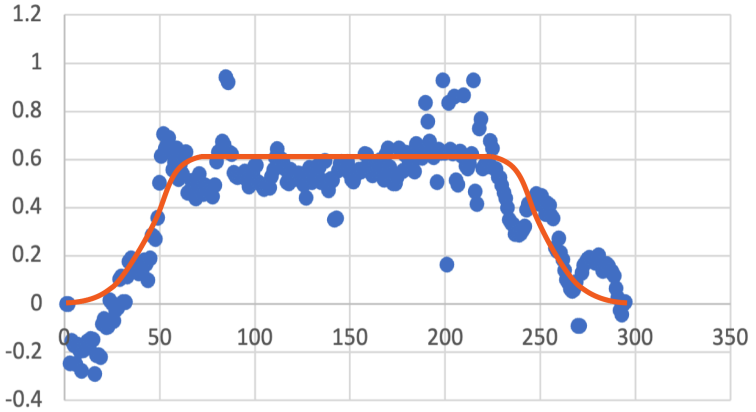


Figure 9a. Graph of horizontal velocity(m/s) vs. time(s) in trial 3



Discussion

Data Interpretation

The overall horizontal displacement (d) vs. time (t) demonstrates the success of the robot when the gradual acceleration profile is implemented. When implemented, the motor started to spin at a low velocity and gradually increased its speed until the maximum velocity was reached. Since the displacement function mentioned in Section I of this essay is defined with a cubic function, the trend line for the data points from Table 2 is a third-degree polynomial trendline, following the mathematical model. The trendline demonstrates a high degree of correlation to the data points, with a coefficient of determination being 0.9983, 0.9943, and 0.9924, respectively. However, the high correlation between the trendline and data could be a result of overfitting, which will be explained in the latter paragraph. The three trials were also rather consistent and aligned with each other, as shown by the equation of the trendline where d is the displacement of the robot (m) and t is the time (s), see Figure 8:

$$d = -0.003t^3 + 0.1035t^2 - 0.698t + 1.5654$$

$$d = -0.0047t^3 + 0.1176t^2 - 0.4755t + 0.367$$

$$d = -0.0085t^3 + 0.1298t^2 - 0.0374t + 0.0382$$

Looking at the velocity (m/s) vs. time (s) graph (Figure 7), one may deduce that the actual velocity output also follows the S-Curve velocity trajectory. As shown by the quadratic piecewise trendline, one could see how the velocity from experimentation increases at a quadratic rate, corresponding to the mathematical calculation performed earlier. Therefore, one can conclude that the S-Curve motion profiling model could be implemented with efficiency on autonomous robotic drivetrains.

Evaluation

Since video footage is analyzed, the frequency of sampling is very high, at a rate of 30 frames per second. As shown in the raw data table (Table 3), there are more than 300 samples collected per trial. Thus, when plotting the points of displacement and velocity against time, the huge number of samples

results in over-fitting of the trendline. For instance, in the displacement vs. time graph for the second trial (Figure 8b), the trendline shows that the object should have an initial decrease in the displacement before increasing which is not achievable or expected. It would be ideal if the program could generate piecewise cubic trendlines to generate a more accurate estimation for the S-curve displacement. A lower change in velocity is also expected when the drivetrain moves further away from the midpoint of the video footage as a result of parallax, as the distance and depth of the camera are shorter than they appear to be. A possible change to minimize the distortion would be to use an acoustic or optical distance sensor instead of taking data from a video camera, which would yield a much more accurate change in displacement, but may result in possible noise.

The time increments per change in velocity are also crucial to the acceleration and deceleration of the robot. Since the program reassesses the displacement of the robot every second, there are only 6 opportunities for velocity changes throughout the entire trajectory. The low refresh rate to change the output velocity in the motors resulted in a parabolic pattern, especially when the robot decelerated (Figure 9b). This pattern is likely to be caused by the friction between the wheels and the floor as they cause internal vibrations when the velocity suddenly decreases, causing the velocity to overshoot.

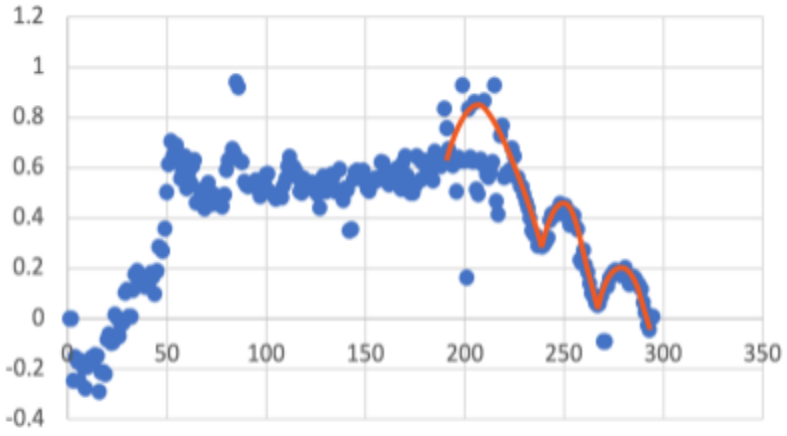


Figure 9b. Robot's velocity plotted against time with identification for the parabolic pattern when velocities change

Future Improvements

Given the target displacement will be changing during the period of the experiment, the position to the positional errors is expected to be greater than the static target displacement value. A better apparatus for this experiment could be to set up the motor upon an air-bearing system so that the motor can move with minimal friction. In this updated apparatus, there would be a laser sensor at the end of the trajectory, recording the motor's relative position throughout the experimentation. This method would create considerably fewer human errors as the object or Target displacement will no longer be moved by hand. Furthermore, by recording the position of the motor, one can derive its velocity throughout the experiment period; compared with the motor encoder recorded values, one can thus find the positional errors due to friction or slippage whenever the motor accelerates or decelerates.

As the velocity increases, the motor is only able to calculate the displacement from the power exported to the motors. This is problematic because the motor has no way of knowing its actual displacement given the friction, slippage, momentum, and torque experienced by the object. Thus, this is not a closed feedback loop, where the computer is operating in the dark, not knowing how much the object physically travelled. An improvement to the methodology could be to utilize the encoder values and calculate the actual displacement given the radius of a wheel or ratios of a gearbox. Since the algorithm is heavily reliant upon the distance travelled, this will significantly improve the accuracy of the velocity output as it can now correctly identify the displacement and determine the phase of the velocity profile.

Real-Life Application

In this experiment, in order to control the distance the robot travels at each trial, the target displacement value was set to be constant, at 15.1. However, in real life, in order to respond to the environment around the vehicle, the vehicle should be re-detecting the target distance prior to the program entering the S-Curve function every time. The sensor must find the distance at a high rate because the distance to the “object” could be changed. For instance, if the vehicle was detecting a wall when parking, the S-Curve equation could be implemented where the sensor only detects in the beginning. However, if a

human were to walk between the vehicle and the wall, the vehicle would not know of a human passing, likely resulting in a car accident. However, by checking the sensor every cycle will allow the vehicle to adjust its velocity with more awareness of its surrounding space.

By implementing the S-Curve equation discussed in this essay, we can optimize for the motion of moving vehicles, robotics arms, and flexible gripping. For instance, the S-curve motion has been proven effective when implemented on live working robotic arms on production lines in tasks such as screwing bolts. The damping motion significantly decreases flexible shocks and internal vibrations, reducing the frequency of stripping in the bolts (Wu et al., 2021).

Conclusion

The mathematical model, virtual simulation, and real-life experimentation presented herein show the coherence and feasibility of this algorithm. The algorithm can be easily translated and applied in any motion controller and actuator. This essay does present a conjecture to the application of S-Curve equations in a wide range of technical applications that require precise control of pressure, speed and passenger comfort, such as robotics and autonomous vehicles.

However, real-life experimentation does demonstrate a few parameters that need to be finetuned for optimal performance. Since momentum is not accounted for in the model when implementing the algorithm on a greater scale, the increased weight of the robot would distort the original acceleration profile. If the refined model could account for the momentum, the S-Curve trajectory could be implemented in even more applications of greater scale, such as public transportation. The systems' energy consumption could be evaluated. It is logical to assume that the energy is saved as efficiently as possible because of the system's smooth and jerk-bounded actions.

References

Bai, Y., Chen, X., Sun, H., & Yang, Z. (2018). Time-Optimal Freeform S-Curve Profile Under Positioning Error and Robustness Constraints. *IEEE/ASME Transactions on Mechatronics*, *23*(4), 1993–2003. https://doi.org/10.1109/TMECH.2018.2835830

Blejan, M. (2020). *Mathematics for Real-Time S-Curve Profile Generator* . https://hidraulica.fluidas.ro/wp-content/uploads/07-25.pdf

Hamm, Karine. "3.4 Acceleration." *Opentextbooks.uregina.ca*, 1 Aug. 2020, opentextbooks.uregina.ca/humanbiomechanics/chapter/2-4-acceleration/.

Meckl, P. H., & Arestides, P. B. (1998a, June 1). *Optimized s-curve motion profiles for minimum residual vibration*. IEEE Xplore. https://doi.org/10.1109/ACC.1998.688324

Mu, Haihua & Zhou, Yunfei & Yan, Sijie & Han, Aiguo. (2009). Third-order trajectory planning for high accuracy point-to-point motion. Frontiers of Electrical and Electronic Engineering in China. 4. 83-87. 10.1007/s11460-009-0017-y.

Nguyen, K. D., Ng, T.-C., & Chen, I-Ming. (2008). On Algorithms for Planning S-Curve Motion Profiles. *International Journal of Advanced Robotic Systems*, *5*(1), 11. https://doi.org/10.5772/5652

Perumaal, S. S., & Jawahar, N. (2013). Automated Trajectory Planner of Industrial Robot for Pick-and-Place Task. *International Journal of Advanced Robotic Systems*, *10*(2), 100. https://doi.org/10.5772/53940

Rappole, B. W. (1992). Minimizing Residual Vibrations in Flexible Systems. *Dspace.mit.edu*, *AITR-1371*. https://dspace.mit.edu/handle/1721.1/6800

Wu, H. (2005). *Application of Improved S-Curve Flexible Acceleration and Deceleration Algorithm in Smart Live Working Robot*. Open Access. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://iopscience.iop.org/article/10.1088/1742-6596/2005/1/012065/pdf

Zou, F., Qu, D., & Xu, F. (2009, December 1). *Asymmetric s-curve trajectory planning for robot*

point-to-point motion. IEEE Xplore. https://doi.org/10.1109/ROBIO.2009.5420482

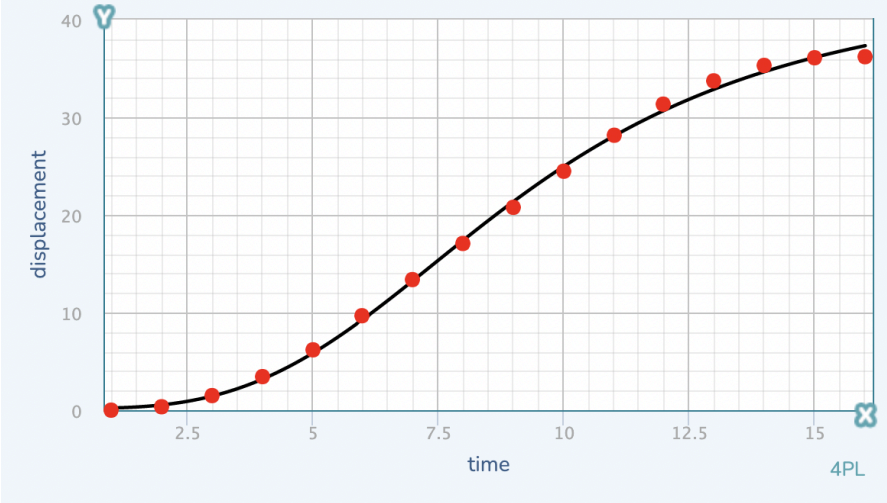
Appendix

Simulation Program in Python:

```
import math

def pow(base, exponent):
    return base ** exponent

def sCurveVel(v, a, x, d, t):
    if x == 0:
        phase = 1
        return 0

    j = math.tan(t * math.pi / 180)
    c0 = a/j
    c1 = v/a
    c2 = d/v

    # Phase 1
    if x < c0:
        phase = 1

        y = j/2 * pow(x, 2)

    # Phase 2
    elif x < c1:
        phase = 2

        y = a * (x - c0/2)

    # Phase 3
    elif x < c1 + c0:
        phase = 3

        y = (-j/2) * pow(c0 + c1 - x, 2) + v

    # Phase 4
    elif x < c2:
        phase = 4

        y = v

    # Phase 5
    elif x < c2 + c0:
        phase = 5

        y = -j/2 * pow(c2 - x, 2) + v

    # Phase 6
    elif x < c2 + c1:
        phase = 6

        y = a * (c2 + c0/2 - x) + v
```

```
    # Phase 7
    elif x < c2 + c1 + c0:
        phase = 7

        y = j/2 * pow(c2 + c1 + c0 - x, 2)

    else:
        phase = -1

        y = None

    return y

vel = 3.7
acc = 0.8
t_displacement = 36.2

correct = [0, 0.35, 1.143, 1.943, 2.743, 3.494, 3.7, 3.7, 3.7, 3.7, 3.684, 3.184,
2.384, 1.584, 0.784, 0.106]

for time in range(16):
    v_out = sCurveVel(vel, acc, time, t_displacement, 35)
    print(v_out)
```

Simulated Data Points of displacement(m) plotted against time(s) with a quartic trendline



Main Program used in testing with single VEX motor in C++

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       sherizhang                                                */
/*    Created:      Wed May 17 2023                                           */
/*    Description:  V5 project                                                */
/*                                                                            */
/*----------------------------------------------------------------------------*/
// ---- START VEXCODE CONFIGURED DEVICES ----
// Robot Configuration:
// [Name]               [Type]        [Port(s)]
// Motor10              motor         10
// RangeFinderA         sonar         A, B
// ---- END VEXCODE CONFIGURED DEVICES ----
#include "vex.h"
#include <cmath>
using namespace vex;
double y;
int phase;
double dis = 0;
double vel = 3.7;
double acc = 0.8;
double t_displacement = 36.2;
double sCurve(double v, double a, double x, double d, double t){
  if (x == 0){
    phase = 1;
    return 0;
  }
    double j = tan(t * M_PI / 180);
    double c0 = a / j;
    double c1 = v / a;
    double c2 = d / v;
    double y = 0.0;
    //Phase 1
    if (x < c0){
      phase = 1;
      y = j/2 * pow(x, 2);
    }

    //Phase 2
    else if (x < c1){
      phase = 2;
      y = a * (x - c0/2);
    }
```

```
    //Phase 3
    else if (x < c1 + c0){
      phase = 3;
      y = (-j/2) * pow(c0 + c1 - x, 2) + v;
    }


    //Phase 4
    else if (x < c2){
      phase = 4;
      y = v;
    }

    //Phase 5
    else if (x < c2 + c0){
      phase = 5;
      y = -j/2 * pow(c2 - x, 2) + v;
    }

    //Phase 6
    else if (x < c2 + c1){
      phase = 6;
      y = a * (c2 + c0/2 - x) + v;
    }
    //Phase 7
    else if (x < c2 + c1 + c0){
      phase = 7;
      y = j/2 * pow(c2 + c1 + c0 - x, 2);
    }


    else{
      phase = -1;
      y = 0;
    }


    return y;
}
int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  t_displacement = RangeFinderA.distance(mm);
    t_displacement = 50;
```

```
  Motor10.spin(forward);
  for (int time = 0; time ≤ 16; time++){
    double v_out = sCurve(vel, acc, time, t_displacement, 35);
    v_out = (v_out / vel) * 100;
    Motor10.setVelocity(v_out,percent);
    Brain.Screen.print(v_out);

    dis = v_out * 1 + dis;
    wait(1,sec);
    Brain.Screen.setCursor(0,0);

  }
  Motor10.setVelocity(0,percent);
}
```

Main Program used in testing with VEX Drivetrain in C++

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       sherizhang                                                */
/*    Created:      Wed May 17 2023                                           */
/*    Description:  V5 project                                                */
/*                                                                            */
/*----------------------------------------------------------------------------*/

// ---- START VEXCODE CONFIGURED DEVICES ----
// Robot Configuration:
// [Name]               [Type]        [Port(s)]
// Motor10              motor         10
// RangeFinderA         sonar         A, B
// Drivetrain           drivetrain    19, 20, 1, 2, 16
// ---- END VEXCODE CONFIGURED DEVICES ----

#include "vex.h"
#include <cmath>

using namespace vex;

double y;
int phase;
double dis = 0;
double vel = 2.38;
double acc = 0.8;
double t_displacement = 15.1;

double sCurve(double v, double a, double x, double d, double t){
 if (x == 0){
```

```
  phase = 1;
  return 0;
}

  double j = tan(t * M_PI / 180);

  double c0 = a / j;
  double c1 = v / a;
  double c2 = d / v;

  double y = 0.0;

  //Phase 1
  if (x < c0){
    phase = 1;

    y = j/2 * pow(x, 2);
  }


  //Phase 2
  else if (x < c1){
    phase = 2;

    y = a * (x - c0/2);
  }


  //Phase 3
  else if (x < c1 + c0){
    phase = 3;

    y = (-j/2) * pow(c0 + c1 - x, 2) + v;
  }

   //Phase 4
  else if (x < c2){
    phase = 4;

    y = v;
  }

  //Phase 5
  else if (x < c2 + c0){
    phase = 5;

    y = -j/2 * pow(c2 - x, 2) + v;

  }

  //Phase 6
  else if (x < c2 + c1){
    phase = 6;
```

```
    y = a * (c2 + c0/2 - x) + v;
  }


  //Phase 7
  else if (x < c2 + c1 + c0){
    phase = 7;

    y = j/2 * pow(c2 + c1 + c0 - x, 2);
  }


  else{
    phase = -1;

    y = 0;
  }

   return y;

}
int main() {
 // Initializing Robot Configuration. DO NOT REMOVE!
 vexcodeInit();

 t_displacement = 15.1;
  Drivetrain.drive(forward);
 //t_displacement = 0.1 * RangeFinderA.distance(mm);

 for (int time = 0; time <= 16; time++){
   double v_out = sCurve(vel, acc, time, t_displacement, 35);

   v_out = (v_out / vel) * 50;
   Drivetrain.setDriveVelocity(v_out,percent);
   Brain.Screen.print(v_out);

   dis = v_out * 1 + dis;

   wait(1,sec);
   Brain.Screen.setCursor(0,0);
   //Brain.Screen.clearScreen();

 }
 Drivetrain.isDone();
 Drivetrain.setDriveVelocity(0,percent);

}
```