TM Assignment 1

Class	TM
Created	@Aug 27, 2020 8:38 PM
Materials	
Reviewed	☐
Type	

Student Information

- Name: Hany Hamed
- Group: BS18-RO
- Big HW1 - Assignment1

Tools

- Python (numpy, Matplotlib)
- LaTex
- Markdown
- Notions

Source

- Lecture about Trajectory Planning
- Lab1 slides

Description

In this assignment, the task is to implement a trajectory planning for a mobile robot that is represented as particle in order to reach the setpoint (goal) as fast as possible. The trajectory planning is under some constraint:

- Maximum Velocity = 1.5 m/s
- Maximum Tangential Acceleration (Max power on the motor) = 10m/s^2
- Maximum Normal Acceleration (Road Adhesion) = 6m/s^2

In order to reach the setpoint as fast as possible, Trapezoidal Velocity profile control will be implemented as it is invented to reach the goal as fast as possible by:

- Having period of time (t_a) to accelerate by the maximum acceleration to reach the maximum velocity.
- Having period of time (t_{vel}) to have zero acceleration, to stay at the maximum velocity.
- Having period of time (t_d) to decelerate by the maximum acceleration to reach the zero velocity.

```
# acceleration time
self.t_a = (self.MAX_VELOCITY_var - abs(self.vel))/self.MAX_ACCELERATION
# deceleration time
self.t_d = (self.vel)/self.MAX_ACCELERATION
self.total_displacement = abs(pos_ref - self.pos)
# total time of acceleration, zero acceleration and deceleration
self.T = (self.total_displacement*self.MAX_ACCELERATION+self.MAX_VELOCITY_var**2)/(self.MAX_ACCELERATION*self.MAX_VELOCITY_va
# total time + current_time_step
self.t_f = self.T + time
```

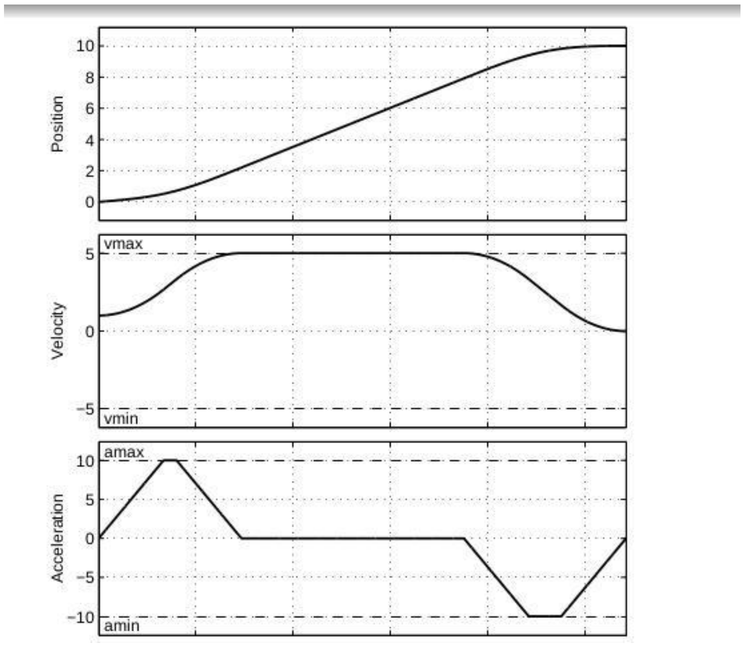

Figure1: Illustration for Trapezoidal Velocity Profile with s-spline.

There are some other planners like: 3^{rd} order polynomial, 5^{th} order polynomial and there are modification that can be implemented with Trapezoidal Velocity profile trajectory planning which is S-spline to make the edges in the planning for velocity much smoother.

Applications

- Such trajectory planners can be found inside motors like Dynamixal.
- Simple trajectory planner for wheeled mobile robotics.

Solution

In the normal planners and the description of the planner inside the lecture from prof. C. Melchiorri, there are no constraints on the tangential acceleration. However, in our problem, there is a constraint on the maximum tangential acceleration as well as on the maximum velocity.

Due to the dependency of the velocity on the tangential acceleration and the curvature which is changing by time, we need to limit the velocity in order not to exceed the constraint on the maximum velocity.

Curvature (k) is calculated as following

$k = |\frac{\ddot{y}(x)}{(1+\dot{y}(x)^2)^{\frac{2}{3}}}|$

And the tangential acceleration is calculated as following

$a_t = v^2 * k$

$a_t^{(max)} = v_{(max)}^2 * k$

$v_{(max)}‘ = min(v_{(max)}, a_t^{(max)}/k)$

And $v_{(max)}‘$ is the new maximum value for the velocity under the constraint of tangential acceleration and this value is being updated with each time step in the simulation.

From the constraint of the maximum tangential and normal acceleration we can get the maximum acceleration from:

$a_{(max)} = \sqrt{a_{t(max)}^2 + a_{n(max)}^2}$

So, the planner now is the same planner in the lecture with limiting the maximum velocity with the original maximum and the function of the curvature each time step.

Note: the simulation is made by choosing a specific dt (delta time) as it is a discrete simulation, which gives us some time steps, less delta time we choose more operation should be done, and more accurate the simulation will be.

$time\_steps = \frac{time_{finish} - time_{start}}{dt}$

The planner is a real-time planner that responds to the changes of the setpoint (update) in real-time as it is checking if the setpoint (position reference) is changed from the previous timestep or not, then it recalculates the timings for deceleration, acceleration and zero acceleration periods. And with each time step, it selects which acceleration should be used based on the delta time and the periods that has been calculated during the setpoint update.

Part of the code for illustration:

Notice that x(t) = t, y(t) = y(x)

```
OM = 3
A = 1
theta_0 = 0.2
dt = 0.001

def motion_equation(t):
    x = t
    y = A*sin(OM*x+theta_0)
    return x,y

total_steps = int(time/dt)

MAX_VELOCITY = 1.5
MAX_ACCELERATION_TANGENTIAL = 10
MAX_ACCELERATION_NORMAL = 6
MAX_ACCELERATION = sqrt(MAX_ACCELERATION_TANGENTIAL**2 + MAX_ACCELERATION_NORMAL**2)

traj_planner = TrapezoidalMotion_mod(max_velocity=MAX_VELOCITY, max_acceleration_normal=MAX_ACCELERATION_NORMAL, max_acceleratio
for s in range(1, total_steps):
    pos = motion_equation(s*dt)
    traj_planner.update(pos[1], s*dt)
    plotter_data_append(traj_planner)

plotter(plot_data)
```

Note: the code has been written in OOP way.

The full class for the planner:

```
class TrapezoidalMotion_mod():
    def __init__(self, max_velocity, max_acceleration_tangential, max_acceleration_normal, initial_pos=0, initial_vel=0, initial_
        self.MAX_ACCELERATION_TANGENTIAL = max_acceleration_tangential
        self.MAX_ACCELERATION_NORMAL = max_acceleration_normal
        self.MAX_ACCELERATION = sqrt(self.MAX_ACCELERATION_NORMAL**2 + self.MAX_ACCELERATION_TANGENTIAL**2)
        self.MAX_VELOCITY = max_velocity

        self.MAX_VELOCITY_var = max_velocity

        self.old_pos = initial_pos
        self.old_pos_ref = initial_pos
        self.old_vel = initial_vel
        self.pos = initial_pos
        self.vel = initial_vel
        self.acc = 0

        self.total_displacement = self.old_pos_ref - self.old_pos

        self.t_a = 0
        self.T = 0
        self.t_f = 0
        self.t_i = 0
        self.delta_time = dt
        self.current_time = 0

        self.curvature = self._calc_curvature()


    def _calc_limited_vel(self):
        velocity = sqrt(self.MAX_ACCELERATION_NORMAL/self._calc_curvature())
        return velocity

    def _calc_max_vel(self):
        # print(self._calc_limited_vel(), self.MAX_VELOCITY)
        return min(self.MAX_VELOCITY, self._calc_limited_vel())


    def _calc_curvature(self):
        x = self.current_time
        dot_f = A*cos(OM*x+theta_0)*OM
        ddot_f = -A*sin(OM*x+theta_0)*OM*OM
        k = abs(ddot_f/(pow((1+dot_f*dot_f),(3/2))))
        return k

    def _calc_normal_acceleration(self):
        return self.vel**2*self._calc_curvature()

    def _calc_tangential_acceleration(self):
        return (self.vel-self.old_vel)/self.delta_time

    def update(self, pos_ref, time):
        self.current_time = time
        self.old_vel = self.vel
        if(self.old_pos_ref != pos_ref):
            self.old_pos = self.pos
            self.old_pos_ref = pos_ref
            self.t_i = time

            self.MAX_VELOCITY_var = self._calc_max_vel()
            self.t_a = (self.MAX_VELOCITY_var - abs(self.vel))/self.MAX_ACCELERATION
            self.t_d = (self.vel)/self.MAX_ACCELERATION
            self.total_displacement = abs(pos_ref - self.pos)
            # self.T = t_a + t_d
            self.T = (self.total_displacement*self.MAX_ACCELERATION+self.MAX_VELOCITY_var**2)/(self.MAX_ACCELERATION*self.MAX_VEL
            self.t_f = self.T + time
        self.calculateTrapezoidalProfile()


    def calculateTrapezoidalProfile(self):
        # if(self.t_i <= self.current_time and self.current_time <= self.t_i+self.t_a):
        if(self.delta_time <= self.t_a):
            # print("+ Acc")
            self.pos = self.pos + self.vel*self.delta_time
            # self.pos = self.old_pos + 1/2*self.MAX_ACCELERATION*((self.current_time - self.t_i)**2)
            self.vel = self.vel + self.MAX_ACCELERATION*self.delta_time
            # print(self.vel)
            self.acc = self.MAX_ACCELERATION

        # elif(self.t_i + self.t_a < self.current_time and self.current_time <= self.t_f - self.t_a):
        elif(self.delta_time > self.t_a and self.delta_time < self.T - self.t_a - self.t_d):
            # print("0 Acc")
            self.pos = self.pos + self.vel*self.delta_time
            # self.pos = self.old_pos + self.MAX_ACCELERATION*((self.current_time-self.t_i-(self.t_a/2))**2)
            self.vel = self.MAX_VELOCITY_var
            self.acc = 0

        # elif(self.t_f - self.t_a < self.current_time and self.current_time <= self.t_f):
        elif(self.delta_time >= self.T - self.t_a - self.t_d and self.delta_time <= self.t_d):
            # print("- Acc")
            self.pos = self.pos + self.vel*self.delta_time
            # self.pos = self.old_pos_ref - (0.5*self.MAX_ACCELERATION*((self.t_f-self.current_time-self.t_i)**2))
            self.vel = self.vel - self.MAX_ACCELERATION*self.delta_time
            self.acc = -self.MAX_ACCELERATION
```

The following figure is the plots for:

- The setpoint position according to the motion equation that is provided
- The curvature changes with respect to the time
- The planned position, you can see that it is barely catching the setpoint, however, if we

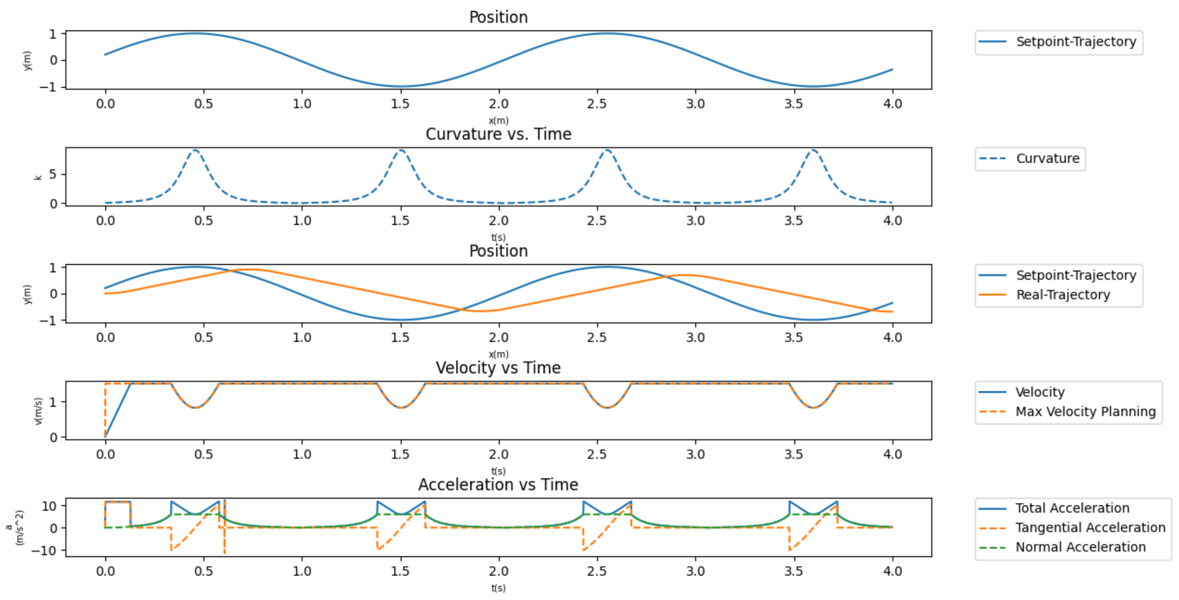

Figure2: Plots related to the motion with trapezoidal velocity profile planner.

For better quality for the plots

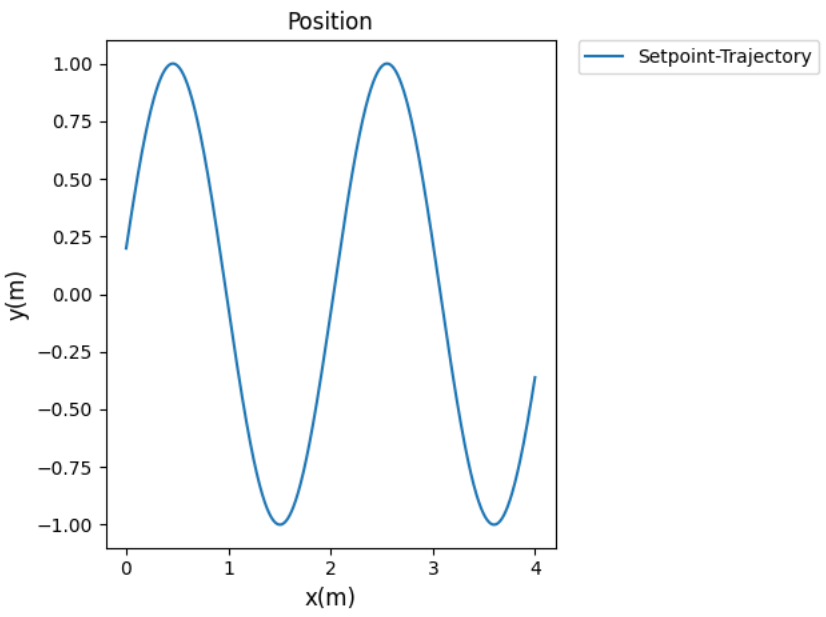

Figure3: Position equation: x(t), y(t)=y(x) they are in meters.

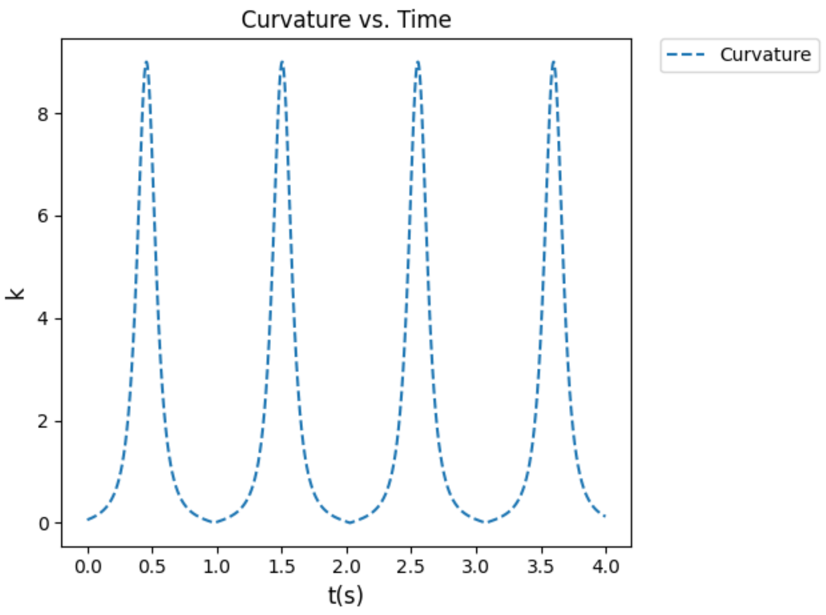

Figure4: Curvature vs. time.

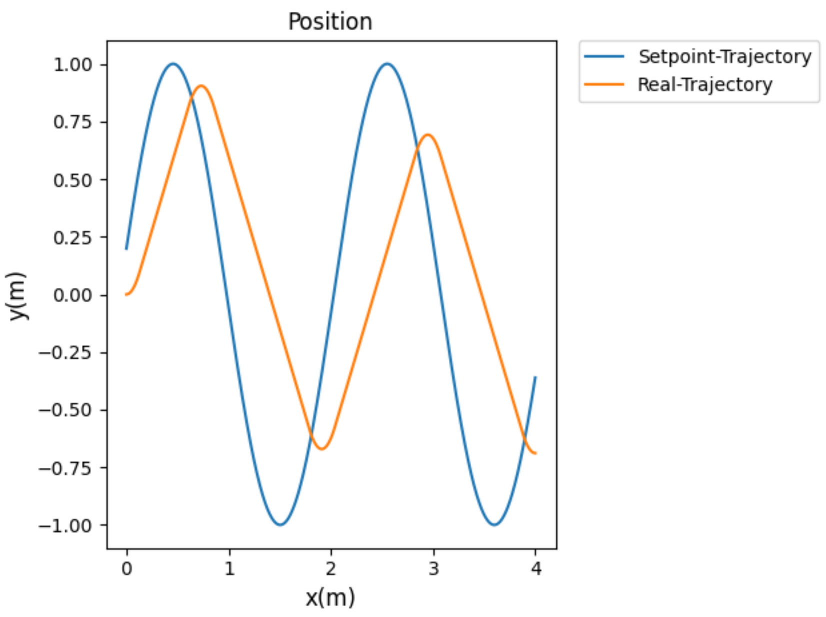

Figure5: Goal positions and Real Trajectory.

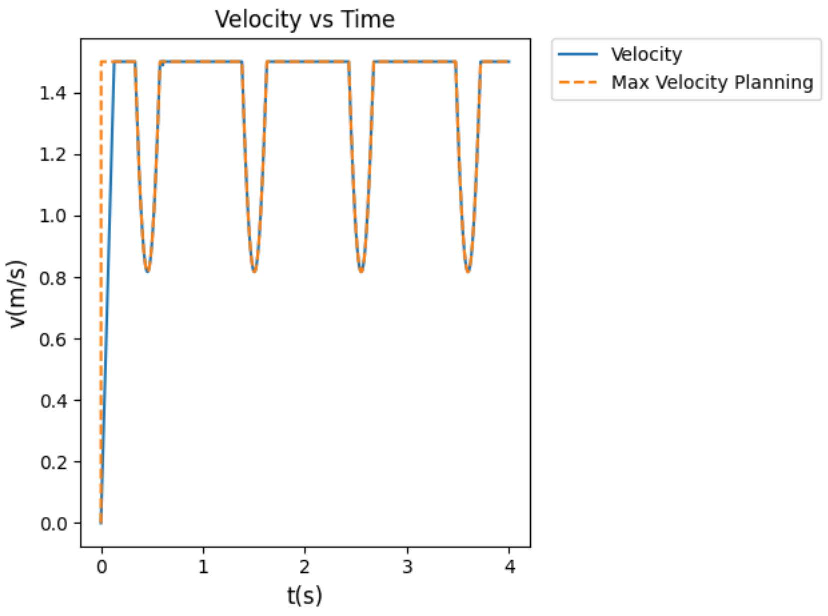

Figure6: Planned Velocity and max velocity calculation based on curvature.

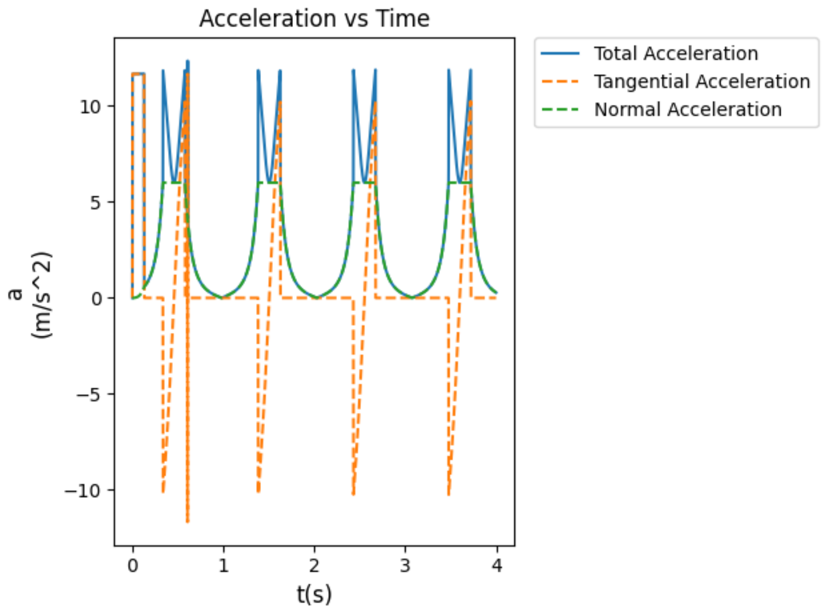

Figure7: Planned acceleration

Note: if the max velocity and the max accelerations increased the vehicle will converge to the setpoint faster and with less error.