A

1. Outline for SCP Parameter tuning

Code work:
https://github.com/EmbersArc/SCvx/tree/master
diffdrive_2d.py

Goal: For each fixed situation, we search for the best parameter.

Situation; Initial condition, Final condition.

There are about three or five types of the SCP algorithm.
The above code is based on the
'SUCCESSIVE CONVEXIFICATION: A SUPERLINEARLY CONVERGENT ALGORITHM FOR NON-CONVEX OPTIMAL CONTROL PROBLEMS'
This paper was submitted to a SIAM journal, but not yet published. There was an error in the proof. Nevertheless, people use the algorithm. Another well-known algorithms are

GuSTO: Guaranteed Sequential Trajectory Optimization via Sequential Convex Programming
https://github.com/UW-ACL/SCPToolbox.jl/tree/csm

Improved Sequential Convex Programming Algorithms for Entry Trajectory Optimization

2. Sequential Convex Programming

Problem:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad g(x)\le 0, h(x)=0. \tag{2.1}$$

eq-2-1

Linearize it by a first-order Talyor series expansion with respect to the given squence $\{x_k\}$, we have

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x_k)+(x-x_k)\nabla f(x_k) \\ \text{subject to} \quad & g(x_k)+(x-x_k)\nabla g(x_k)\le 0 \\ & h(x_k)+(x-x_k)\nabla h(x_k)=0 \\ & \|x-x_k\|\le \epsilon_k, \end{aligned} \tag{2.2}$$

eq-2-2

where the last constraint is called trust region with $\epsilon_k \ge 0$. Note that $f(x_k)$ in problem can be erased.

Consider the case that constraints are removed, then problem is

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \;& f(x_k) + (x - x_k)\nabla f(x_k) \\ \text{subject to} \quad & \|x - x_k\| \leq \epsilon_k. \end{aligned} \tag{2.3}$$

We can know that the update rule of above problem is

$$x_{k+1} = x_k - \epsilon_k \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}, \tag{2.4}$$

which is a form of GD with stepsize $\frac{\epsilon_k}{\|\nabla f(x_k)\|}$.

But in real-world, problems we need to solve are usually having many constraints. In (2.1) and (2.2), maybe we get the artificial infeasibility which means

$$\begin{aligned} g(x) \leq 0 \quad & \text{has a solution, but} \\ g(x_k) + (x - x_k)\nabla g(x_k) \leq 0 \quad & \text{has no solution.} \end{aligned} \tag{2.5}$$

Similarly,

$$\begin{aligned} h(x) = 0 \quad & \text{has a solution, but} \\ h(x_k) + (x - x_k)\nabla h(x_k) = 0 \quad & \text{has no solution.} \end{aligned} \tag{2.6}$$

So, we use the slack variable $v \in \mathbb{R}^n$ called virtual control such that

$$h(x_k) + (x - x_k)\nabla h(x_k) + v = 0 \tag{2.7}$$

and the problem is recontructed by

$$\begin{aligned} \min_{x\in\mathbb{R}^n, v} \quad & f(x_k) + (x - x_k)\nabla f(x_k) + \eta_k \|v\|^2 \\ \text{subject to} \quad & g(x_k) + (x - x_k)\nabla g(x_k) \leq 0 \\ & h(x_k) + (x - x_k)\nabla h(x_k) = 0 \\ & \|x - x_k\| \leq \epsilon_k. \end{aligned} \tag{2.8}$$

2.1. **Sequential convex programming for the optimal control problems.** Consider the problem

$$\begin{aligned} \min_u \quad & \int_0^T g(x(t), u(t))dt \\ \text{subj. to} \quad & x'(t) = f(x(t), u(t)) \\ & x(0) = x_0, \quad x(T) = x_f \\ & \text{Other constraints of } (x(t), u(t)) \end{aligned} \tag{2.9}$$

We linearize the problem near $(x_k(t), u_k(t))$ where $k \geq 0$ denotes the outer iteration number.

Assume that g is already a convex function. We linearize the problem as

$$\begin{aligned}
&\min_{u(t)} \int_0^T g(x(t),u(t))dt + w_\nu\|\nu(t)\|_1 + \omega_x\|\delta_x(t)\|_2 + \omega_u\|\delta_u(t)\|_2 \\
&\text{subj. to } \dot{x}(t) = A(t)x(t) + B(t)u(t) + \Sigma(t) + z(t) + \nu(t) \\
&\qquad\quad \text{(Linearized) Constraints of } (x(t),u(t)) \\
&\qquad\quad \|x(t) - x_{last}(t)\| \leq \delta_x(t) \\
&\qquad\quad \|u(t) - u_{last}(t)\| \leq \delta_u(t).
\end{aligned} \tag{2.10}$$

Here $\nu(t)$ is a virtual control and

$$\begin{aligned}
A(t) &= \nabla_x f(x_{last}(t), u_{last}(t)) \\
B(t) &= \nabla_u f(x_{last}(t), u_{last}(t)) \\
\Sigma(t) &= f(x_{last}(t), u_{last}(t)). \\
z(t) &= -\nabla_x f(x_{last}(t), u_{last}(t))x_{last}(t) - \nabla_u f(x_{last}(t), u_{last}(t))u_{last}(t)
\end{aligned}$$

'Compute-matrixes' calls the functions 'Compute-formula' and 'Compute-transition'.

'Compute-formula' calculate the matrices $A(t)$, $B(t)$, $\Sigma(t)$, and $z(t)$.

'Compute-transition' calculate the matrices $\bar{A}(t)$, $\bar{B}(t)$, $\bar{C}(t)$, $\bar{\Sigma}(t)$, and $\bar{z}(t)$.

Improved Sequential Convex Programming Algorithms for Entry Trajectory Optimization

2.2. **Linearization of the dynamic.** We consider the dyanmic

$$\dot{x}(t) = f(x(t), u(t)), \quad t \in [0,T]. \tag{2.11}$$

Here $x(t) \in \mathbb{R}^{N_x}$ is state and $u(t) \in \mathbb{R}^{N_u}$ is control. The cost is given by a function of $x(t)$ and $u(t)$.

We linearize the above equation using

$$\begin{aligned}
&f(x(t),u(t)) \\
&= f(x(t) - x_{last}(t) + x_{last}(t), u(t) - u_{last}(t) + u_{last}(t)) \\
&= f(x_{last}(t), u_{last}(t)) + \nabla_x f(x_{last}(t), u_{last}(t))(x(t) - x_{last}(t)) \\
&\quad + \nabla_u f(x_{last}(t), u_{last}(t))(u(t) - u_{last}(t)) + O(diff^2) \\
&\simeq \nabla_x f(x_{last}(t), u_{last}(t))x(t) + \nabla_u f(x_{last}(t), u_{last}(t))u(t) \\
&\quad + f(x_{last}(t), u_{last}(t)) - \nabla_x f(x_{last}(t), u_{last}(t))x_{last}(t) - \nabla_u f(x_{last}(t), u_{last}(t))u_{last}(t) \\
&=: A(t)x(t) + B(t)u(t) + z(t),
\end{aligned} \tag{2.12}$$

where $diff = \|x(t) - x_{last}(t)\|$ and

$$
\begin{aligned}
A(t) &= \nabla_x f(x_{last}(t), u_{last}(t)) \\
B(t) &= \nabla_u f(x_{last}(t), u_{last}(t)) \\
z(t) &= f(x_{last}(t), u_{last}(t)) - \nabla_x f(x_{last}(t), u_{last}(t))x_{last}(t) - \nabla_u f(x_{last}(t), u_{last}(t))u_{last}(t).
\end{aligned}
$$

We now have

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + z(t). \tag{2.13}$$

To avoid the artificial infeasibility, we use a slack variable $\nu(t)$ to modify the equation as

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + z(t) + \nu(t). \tag{2.14}$$

And we add the following additional cost

$$w_\nu \int_0^T \|\nu(t)\| dt. \tag{2.15}$$

2.3. **Time Discretization for the linearized dynamic.** Consider the following equation

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + z(t). \tag{2.16}$$

Here $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{n\times m}$. Also, $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$. For $t \in [t_i, t_{i+1}]$ we consider

$$
\begin{aligned}
u(t) &= \beta^{-1}(t)u_i + \beta^{+}(t)u_{i+1} \quad \text{for } t \in [t_i, t_{i+1}]. \\
&= \frac{1}{\Delta t}(t_{i+1} - t)u_i + \frac{1}{\Delta t}(t - t_i)u_{i+1}.
\end{aligned} \tag{2.17}
$$

On each interval $[t_i, t_{i+1}]$, we approximate the above problem by

$$\dot{x}(t) = A_i x(t) + B_i(t)u(t) + z(t) \quad \forall\ t \in [t_i, t_{i+1}], \tag{2.18}$$

where $A_i = A(t_i)$ and $B_i = B(t_i)$. The error might be $O(h^2)$. The above problem is solved exactly as

$$
\begin{aligned}
x(t) = e^{(t-t_i)A_i}x(t_i) + u_i \int_{t_i}^{t} \beta^{-}(\tau)e^{(t-\tau)A_i}B_i\, d\tau + u_{i+1} \int_{t_i}^{t} \beta^{+}(\tau)e^{(t-\tau)A_i}B_i d\tau \\
+ \int_{t_i}^{t} e^{(t-\tau)A_i}z(\tau)d\tau.
\end{aligned} \tag{2.19}
$$

Hence,

$$x(t_{i+1}) = A_i x(t_i) + B_i^{-}u_i + B_i^{+}u_{i+1} + z_i, \tag{2.20}$$

eq-1-1

where

$$
\begin{aligned}
A_i &= e^{(t_{i+1}-t_i)A} \\
B_i^{-} &= \int_{t_i}^{t_{i+1}} \beta^{-}(\tau)e^{(t_{i+1}-\tau)A}B_i\, d\tau, \\
B_i^{+} &= \int_{t_i}^{t_{i+1}} \beta^{+}(\tau)e^{(t_{i+1}-\tau)A}B_i\, d\tau, \\
z_i &= \int_{t_i}^{t_{i+1}} e^{(t_{i+1}-\tau)A}z(\tau)d\tau.
\end{aligned} \tag{2.21}
$$

We add a slack variable ν_i to (2.20) to avoid the artificial infeasibility from the linearization. That is,

$$x(t_{i+1}) = A_i x(t_i) + B_i^- u_i + B_i^+ u_{i+1} + z_i + \nu_i. \tag{2.22}$$

We add the following weight to the cost

$$w_\nu \sum_{i=0}^{K} \|\nu_i\|_2. \tag{2.23}$$

3. Sequential Convex Programming (SCP) with Warm-start by using Deep Neural Network

1. Problem Setup. We seek a control trajectory $u_0, \ldots, u_N$ and corresponding state trajectory $x_0, \ldots, x_N$ that drives the system from $x_0 = x_{\text{init}}$ to $x_N = x_{\text{final}}$ over a free time T_f, divided into N intervals of length $\Delta T = T_f/N$.

The basic minization problem of Optimal Control is

$$\begin{aligned} \min_{x,u,T_f} \; & T_f + \int_0^{T_f} \ell(x(t), u(t))dt \\ \text{subj.to } & \dot{x} = f(x,u), \\ & x(0) = x_{\text{init}}, \\ & x(T_f) = x_{\text{final}}, \\ & u_{min} \leq u(t) \leq u_{max} \end{aligned} \tag{3.1}$$

where R is an Identity matrix.

2. Detailed SCP Procedure for general case. This procedure only describes the case with initial, finial state and state dynamics and linear inequality constraints. At each iteration k, we perform the following steps.

Step 1: Dynamics Linearization (ZOH). We linearize the dynamics:

$$x_{i+1} \approx A_i x_i + B_i u_i + T_f s_i + z_i + v_i^{(c)}$$

where $A_i = \frac{\partial f}{\partial x}$, $B_i = \frac{\partial f}{\partial u}$ evaluated at $(x_i^{(k)}, u_i^{(k)}, T_f^{(k)})$, and $v_i^{(c)}$ is a virtual control term.

Step 2: min-max Scaling. We normalize state, control, and the final time:

$$\tilde{x}_i = S_x^{-1}(x_i - s_x), \quad \tilde{u}_i = S_u^{-1}(u_i - s_u), \quad \sigma = S_\sigma^{-1} T_f$$

with $S_x = \text{diag}(x_{\max} - x_{\min})$, $s_x = x_{\min}$, and similarly for S_u and s_u. The time scaling factor S_σ is typically set from the initial guess, $S_\sigma = T_f^{\text{init}}$. Therefore the initial value of σ for iteration is 1.

Step 3: Convex Subproblem Formulation.

$$
\begin{aligned}
\min_{\tilde{x}_i,\tilde{u}_i,\sigma,v_i^{(c)}} J =& S_\sigma \sigma + \sum_{i=0}^{N} w_c \cdot \ell(S_x\tilde{x}_i + s_x, S_u\tilde{u}_i + s_u) + \sum_{i=0}^{N-1} w_{vc} \cdot \|v_i^{(c)}\|_1 \\
&+ \sum_{i=0}^{N} w_{tr} \cdot \left(\|\tilde{x}_i - \hat{x}_i^{(k)}\|^2 + \|\tilde{u}_i - \hat{u}_i^{(k)}\|^2\right) + w_{tr} \cdot (\sigma - \hat{\sigma}^{(k)})^2 \\
&+ \sum_{i=0}^{N-1} w_{\text{rate}} \cdot \|\tilde{u}_{i+1} - \tilde{u}_i\|^2 \\
\text{subject to} \quad & S_x\tilde{x}_0 + s_x = x_{\text{init}}, \\
& S_x\tilde{x}_N + s_x = x_{\text{final}}, \\
& S_x\tilde{x}_{i+1} + s_x = A_i(S_x\tilde{x}_i + s_x) + B_i(S_u\tilde{u}_i + s_u) + (S_\sigma\sigma)s_i + z_i + v_i^{(c)}, \quad \forall i \in \{0,\ldots,N-1\} \\
& u_{\min} \le S_u\tilde{u}_i + s_u \le u_{\max}, \quad \forall i \in \{0,\ldots,N\}
\end{aligned}
\tag{3.2}
$$

where $\hat{x}_i$, $\hat{u}_i$, $\hat{\sigma}$ is the value of σ from the previous iteration k. Also, we use $x_i = S_x\tilde{x}_i + s_x$ and $u_i = S_u\tilde{u}_i + s_u$ in the original cost function $\sum_{i=0}^{N} \ell(x_i, u_i)$ since we want to preserve the physical meaning of the cost and manipulate the importance by modifying parameter w_c.

Step 4: Inverse Scaling of solutions. After solving, we perform inverse scaling:

$$x_i^{(k+1)} = S_x\tilde{x}_i^* + s_x, \quad u_i^{(k+1)} = S_u\tilde{u}_i^* + s_u, \quad T_f^{(k+1)} = S_\sigma\sigma^*$$

Step 5: Nonlinear Forward Simulation. The objective of this step is to compute the state trajectory x^{fwd} by applying the optimized control sequence $u^{(k+1)}$ and final time $T_f^{(k+1)}$ to the true nonlinear dynamics $\dot{x} = f(x, u)$, starting from x_{init}. This is essential for Step 6, which compares the prediction from the linearized model ($x^{(k+1)}$) with the result from the nonlinear model (x^{fwd}) to check for convergence. The process involves a sequential integration loop.

1. Simulation Setup.

- Inputs: The control sequence $u^{(k+1)} = [u_0^{(k+1)}, \ldots, u_N^{(k+1)}]$, the final time $T_f^{(k+1)}$ from the previous step, and the given initial state x_{init}.
- Time Step Calculation: The duration of each discrete interval is calculated as $\Delta t = T_f^{(k+1)}/N$.
- Trajectory Initialization: A variable x^{fwd} is prepared to store the simulated trajectory, initialized with $x_0^{\text{fwd}} = x_{\text{init}}$.

2. Sequential Integration Loop. The simulation proceeds by iterating through each interval $i = 0, \ldots, N-1$:

- ODE Solver Call: An ODE solver, such as `scipy.integrate.solve_ivp`, is used to propagate the system forward in time by Δt, starting from the current state x_i^{fwd}.

- Control Input Application(ZOH): The control input is held constant at $u_i^{(k+1)}$.

$$u(t) = u_i^{(k+1)} \quad \text{for } t \in [t_i, t_{i+1}]$$

- State Update: The result of the integration at the end of the interval, $x(t_{i+1})$, becomes the next state in the simulated trajectory, x_{i+1}^{fwd}.

3. Final Result. Upon completion of the loop, the full nonlinear simulated trajectory $x^{\text{fwd}} = [x_0^{\text{fwd}}, x_1^{\text{fwd}}, \ldots, x_N^{\text{fwd}}]$ is obtained. This trajectory is then used in Step 6 to check for Boundary Consistency.

Step 6: Convergence Criteria. We check for convergence by verifying:

(1) Boundary consistency:

$$\max_i \|x_i^{(k+1)} - x_i^{\text{fwd}}\| < \text{tol}_{bc}$$

(2) Virtual control tolerance:

$$\sum_i \|v_i^{(c)}\|_1 / w_{vc} < \text{tol}_{vc}$$

(3) Trust region:

$$\sum_i \left(\|\tilde{x}_i^{(k+1)} - \tilde{x}_i^{(k)}\|^2 + \|\tilde{u}_i^{(k+1)} - \tilde{u}_i^{(k)}\|^2 \right) / w_{tr} < \text{tol}_{tr}$$

Step 7: Output. If the criteria of Step 6 satisfied after some iterations, the algorithm returns:

- x^*, u^*, T_f^*: optimal state, control trajectories and optimal final time.
- J: total cost.

3. SCP Warm-start by DNN (Unicycle case).

- In general nonlinear optimal control problems, there is no need to provide an initial control sequence explicitly—control inputs are optimized along with the trajectory.
- However, in SCP, since each iteration involves linearizing the dynamics and constraints around a reference trajectory, an initial guess of the control sequence is essential.
- Warm-starting can provide an initial control sequence that lies close to the optimal region of the objective function.
- Without a warm-start, the optimization may begin from a poor initial guess, which increases the risk of convergence failure or even infeasibility.

Procedure for Deep Learning.

- We use deep learning to generate suitable initial guesses for SCP[**?**].
- To do this, we first solve the SCP problem multiple times using a variety of initial state inputs.
- For each case, we store the resulting optimal control sequence.
- Using this data, we train a neural network that maps an initial guess to the corresponding optimal control sequence.

- Once trained, the network can quickly produce a near-optimal initial guess that improves SCP convergence.

Unicycle dynamics. Let the state and control be

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}, \quad u = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where

- p_x: position along the x-axis (horizontal position),
- p_y: position along the y-axis (vertical position),
- θ: heading angle (orientation) of the robot in radians,
- v: linear velocity in the direction of the heading,
- ω: angular velocity (rate of change of heading).

Then the continuous-time dynamics are:

$$\dot{x} = f(x, u) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}$$

The basic minization problem is

$$\begin{aligned} \min_{x,u,T_f} \ & T_f + \int_0^{T_f} u(t)^T R u(t) dt \\ \text{subj.to } & \dot{x} = f(x, u), \\ & x(0) = x_{\text{init}}, \\ & x(T_f) = x_{\text{final}}, \\ & \begin{bmatrix} -2 \\ -2 \end{bmatrix} \leq u(t) \leq \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \end{aligned} \tag{3.3}$$

where R is an Identity matrix.

Data Generation.

- Initial and Final states:
  $x_{\text{init}} = np.zeros(3)$
  $x_{\text{init}}[0] = -1.0 + np.random.normal(0, 0.5)$
  $x_{\text{init}}[1] = -2.0 + np.random.normal(0, 0.5)$
  $x_{\text{init}}[2] = 0$
  $x_{\text{final}} = np.zeros(3)$
  $x_{\text{final}}[0] = 2.0 + np.random.normal(0, 0.5)$
  $x_{\text{final}}[1] = 2.0 + np.random.normal(0, 0.5)$
  $x_{\text{final}}[2] = 0$

- Initial trajectory is given as:
  $x_0 = np.zeros((N+1, ix))$
  for j in range$(N+1)$:
  $x_0[j] = (N-j)/N * x_{\text{init}} + j/N * x_{\text{final}}$
- Initial control sequence is given as:
  $u_0 = np.zeros((N+1, iu))$
- By solving SCP problem for 20000 times, we get the dataset of 20000 pair of $([x_{\text{init}}, x_{\text{final}}]^T, v^*)$ and $([x_{\text{init}}, x_{\text{final}}]^T, T_f^*)$.
- By experiment, we check the fact that learning ω^* and applying it to initial control sequence does not improve the algorithm, the calculation time rather increases.
- We build two Deep Neural networks which has $[x_{\text{init}}, x_{\text{final}}]^T$ as an input and v^*, T_f^* as an output.

Results.
Control Learning(v^*). This is the example of the result of control learning v^*. We put this DNN result to SCP algorithm as an initial control sequence.

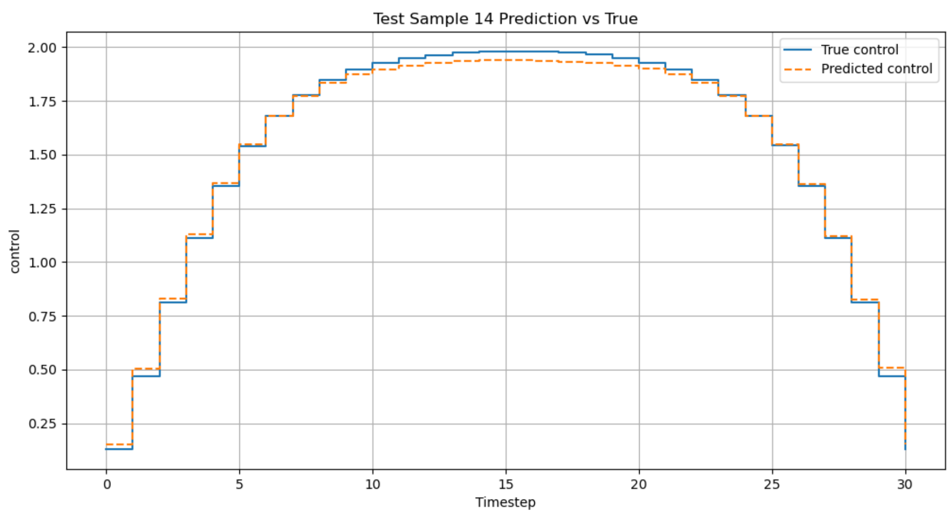

FIGURE 1. Example of learning v^*

Warm-start Result. start= $[-0.483, -2.205]$, goal= $[1.76, 1.92]$
$T_f^* = 13.1$
iteration: Original SCP= 11, SCP with Warm start= 6
time: Original SCP= 2.5sec, SCP with Warm start= 1.3sec

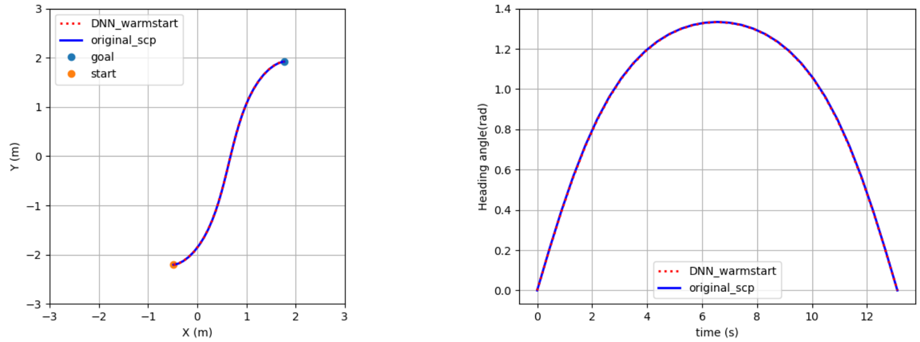

FIGURE 2. State trajectory and heading angle

Sh

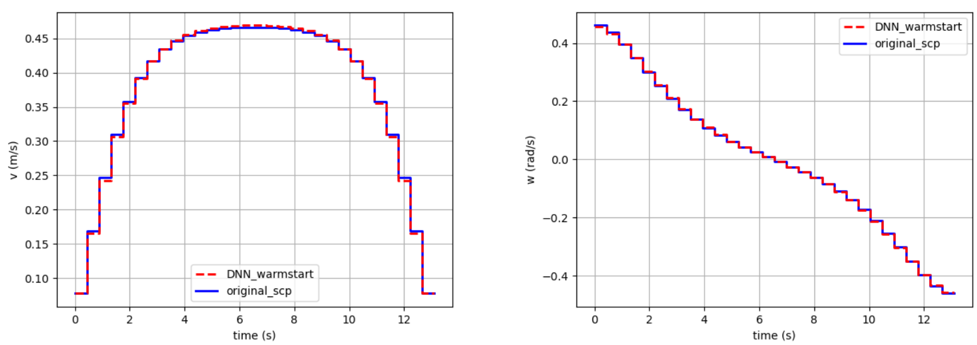

FIGURE 3. Linear and Angular Velocity

LA