# Dynamics Simulation in Robot Systems

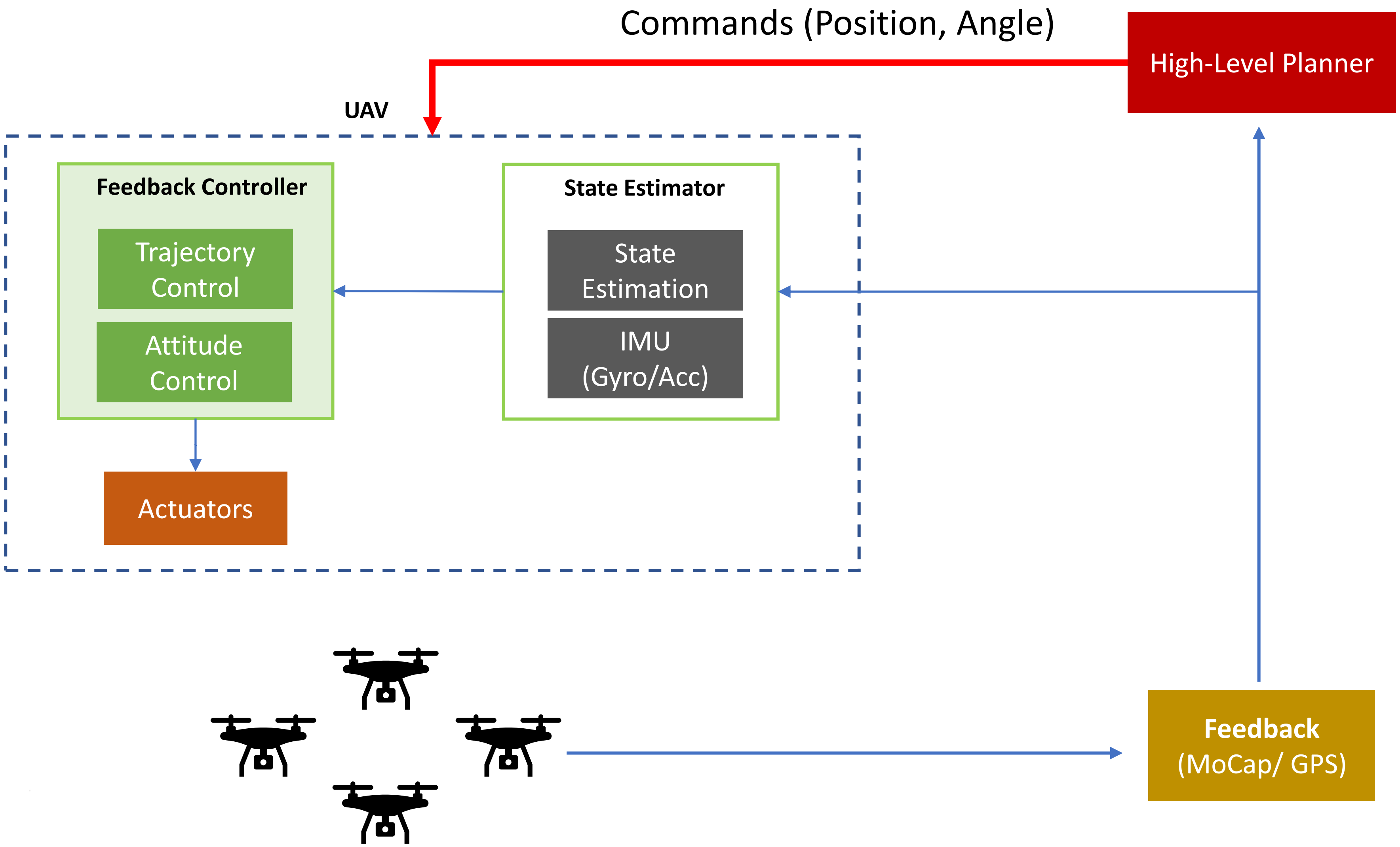
Malintha Fernando
Department of Intelligent Systems Engineering
Indiana University

# Outline

- Unmanned Aerial Vehicles (UAV)
- Simulation Environments
  - Robot Operating System
  - Gazebo
- Definitions
- Dynamics and Control
  - Differential Driven Dynamics
  - Equations of Motion
  - Numerical Integration
  - Differential Flatness
  - Feedback Control

# Unmanned Aerial Vehicles (UAVs)

# How does it work?




# Definitions

- **Kinematics**: The motion of rigid bodies (displacement, velocity etc.) regardless of the actual forces acting on the system such as friction E.g.: a trajectory of a particle in a frictionless plane.
- **Dynamics**: The motion of rigid bodies (displacement, velocity etc.) without referring to the actual forces such as friction. E.g.: how would the trajectory change if we apply friction?
- **Control**: Manipulating the actuators of a physical system to result in the desired behavior by accounting for the dynamics. E.g: changing the angle of the plane, so the particle would stay in place.
- **Degrees of Freedom:** The number of independent parameters we can use to define the configuration (State) of a system. E.g.: The particle has 2 DoF (X,Y) with respect to the plane.

# Quadrotor Dynamics (& Definitions

- A particle in the space has **3 Degrees of Freedom (DOF)**: {X, Y, Z}.
- A rigid body in the space has **6 Degrees of Freedom (DOF)** : {Roll, Pitch, Yaw, X, Y, Z}.
- Any mechanical system that can control each DOF **independently** is known as a **fully-actuated** system.
- Any mechanical system that has lesser independent actuation variables than DOF is known as an **underactuated** system.
- A quadrotor has an **underactuated**, **non-linear** dynamic model.

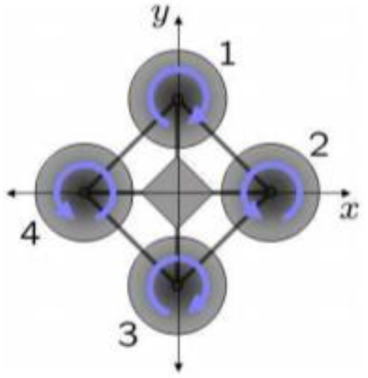



Figure 2.3: Direction of propeller's rotations.

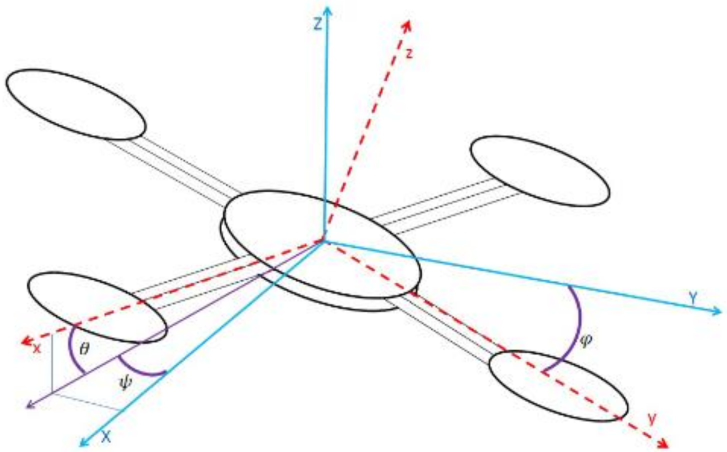


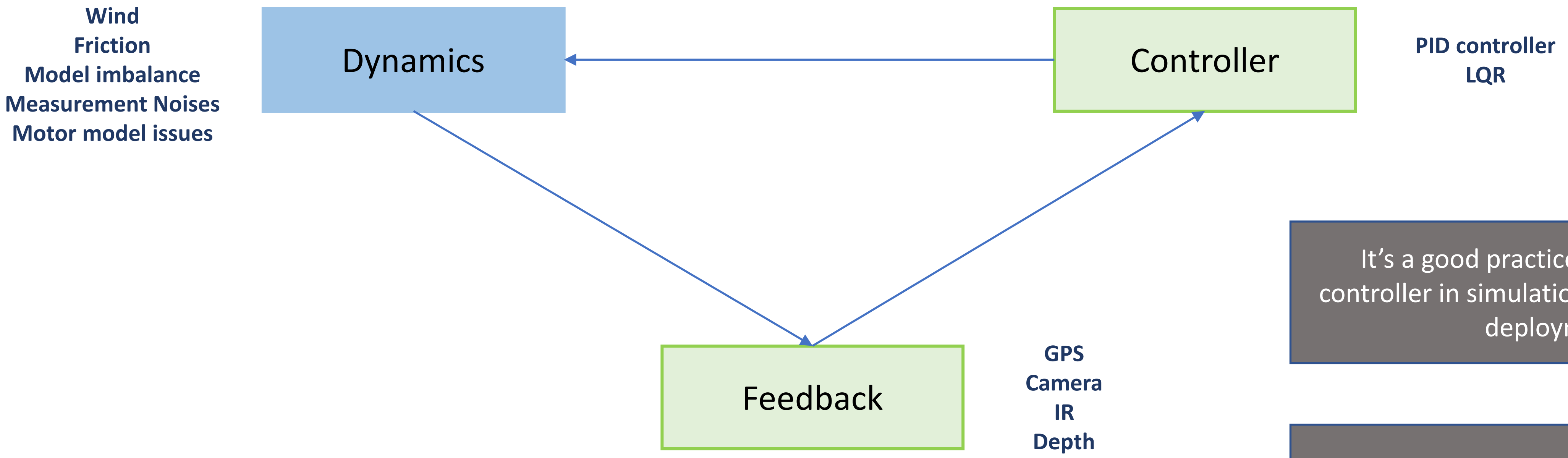
Figure 2.8: Euler Angles.

# However,

Fully actuated UAV with 6 rotors

Fully actuated UAV with only 2 rotors

# System Design




It's a good practice to validate your controller in simulation prior to real-world deployments.

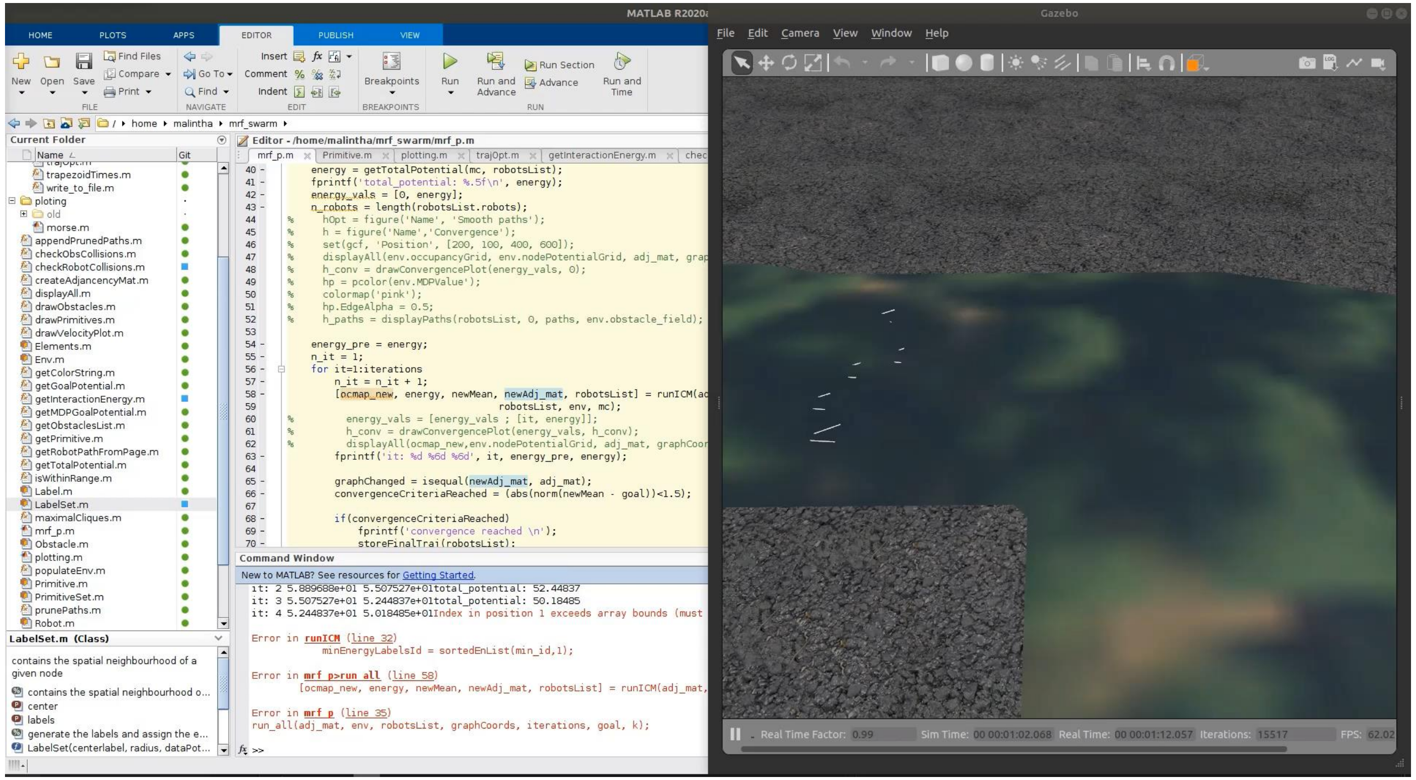
We will study simulating dynamics of a system in the next few slides.

**A typical feedback controller loop for a robotic system**

# Simulation Platforms in Robotics

- Physics simulation engines:
  - Gazebo (C++, Python)
  - PyGame (Python)
  - MATLAB Simulink (Matlab)

- Other Visualization Platforms (No Physics)
  - Rviz (C++, Python)
  - Matlab
  - Any other plotting tool




A UAV team simulation with Gazebo

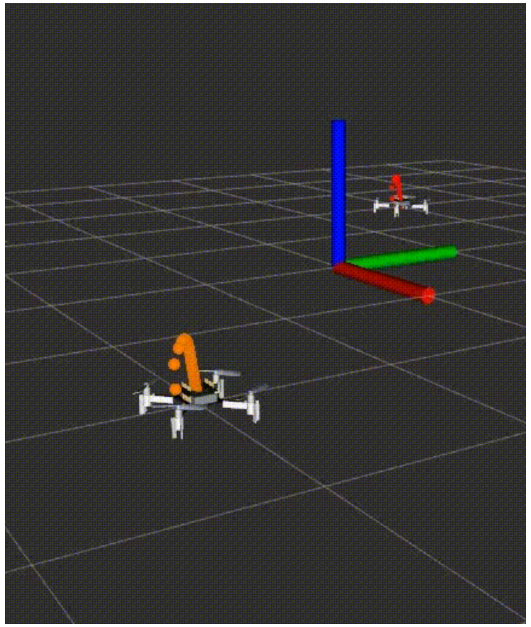
Simulation engines also provide the feedback on the robot state and simulate environmental disturbances.

# Simulating Dynamic Systems

- Identify the system model (Manufacturer given or experimentally)
  - Dimensions
  - weight
  - moment of inertia
  - ~~Motor constants~~
  - ~~Thrust coefficients etc.~~

**Actuator Dynamics (Only for real-world deployments)**

- Develop the kinematic equations.
- Combine with the dynamics.
- Implement the dynamic model.
  - Integrate the equations throughout time
- Validate the system using known actuator commands



**Drones recovering from upside-down initialization**

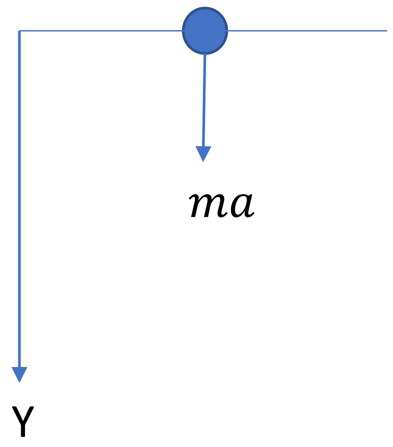
# Simulating a Particle

- Consider a particle in space. Let's write the differential equations for its position and velocity assuming a constant acceleration.

- First, position and velocity changes over time. So, we write:

$v(t)$ : $velocity\ as\ a\ function\ of\ time$
$y(t)$: position as a function of time
$a(t) = a(0)$ Constant Acceleration

- Goal is to write the differential equations for this system, so we can simulate by numerical integration.




Initial Conditions:
$v(0) = 0, y(0) = 0, a(0)$

# Simulating a Particle

- First consider the change of position over an infinitesimal time period $dt$

$$v(t+dt) - v(t) = dt * a$$

$$\frac{v(t+dt) - v(t)}{dt} = a$$

- Taking the limit on both sides:

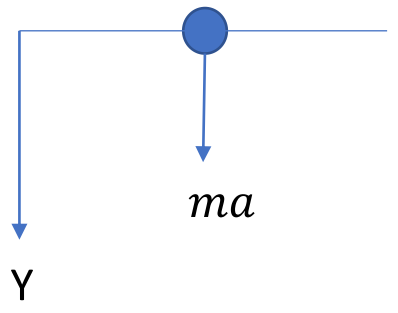
$$\lim_{dt \to 0} \left\{ \frac{v(t+dt) - v(t)}{dt} \right\} = a$$

The definition of the time derivative.

$$\dot{v} = a$$

- Do the same for the position.

$$\dot{y} = v$$




Initial Conditions:
$v(0) = 0, y(0) = 0, a(0)$

# Simulating a Particle

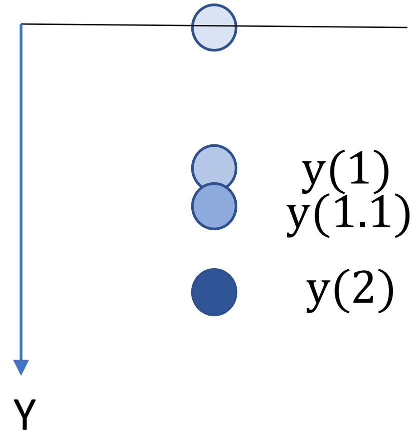
Initial Conditions:

$v(0) = 0,$
$\mathrm{y}(0) = 0,$
$a(0) = g$

- Consider a particle in space, assuming the only force working on it is gravity, we can write the following equations for the downward motion.

$$\dot{v} = g$$
$$\dot{y} = v$$

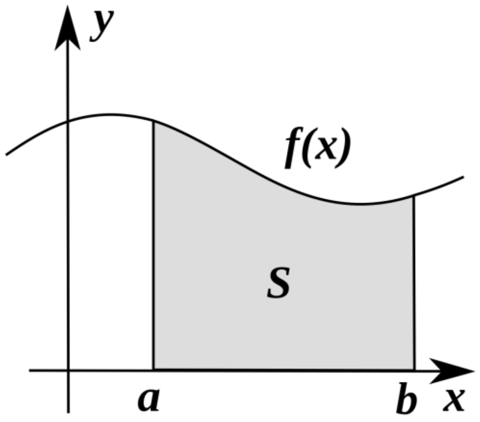
- But these equations only provide us how the system would change over an infinitesimal time.
- To see how these small steps accumulate over time, we have to integrate the system over a desired period.
- As the initial conditions change, the system behavior would also change.
- For that, we use Numerical integration.



# Simulating a Particle

- The basic idea of numerical integration is to calculate the finite integral of a function between a given period accurately.

$$\int_a^b f(x)\,dx$$




- Many algorithms have been proposed for this (Eulers', Reimann Sum, Quadrature methods) in numerical analysis.

- We will just use the inbuilt integration functions from scientific computing packages. E.g.: odeint from scipy, ODE45 from MATLAB, gsl_odeiv2 in GNU Sciene Library (GSL).

- Typically, we will have a single function inside which we include all our differential equations. And we stack all the LHS values on top of each other in an array.

- A reference to this function is then passed to the solver, which uses our differential equations to perform the integration.

# Simulating a Particle

New system configuration are recursively passed inside.

```
def f(y, t):
    """this is the rhs of the ODE to integrate, i.e. dy/dt=f(y,t)"""
    y_, v_ = y

    fdot = [v_, 9.82]
    return fdot

y0 = [0, 0]             # initial value
a = 0                # integration limits for t
b = 2

t = N.arange(a, b, 0.01)   # values of t for
                           # which we require
                           # the solution y(t)
f_val = odeint(f, y0, t)   # actual computation of y(t)
```

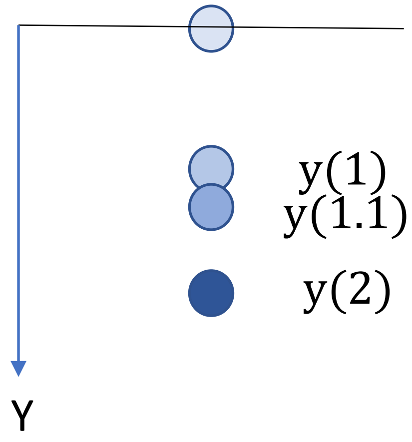
Code your equations here by reading from the array.

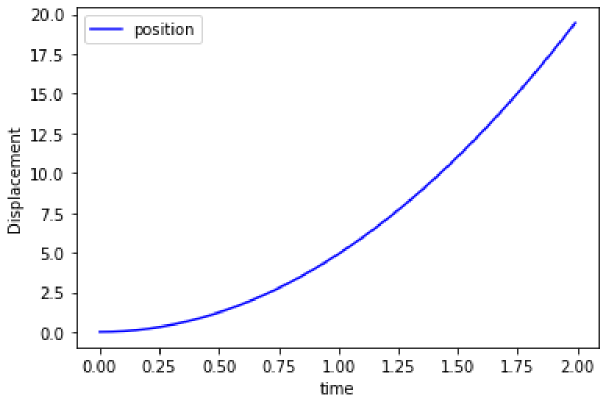
Stack all the LHS values in an array.

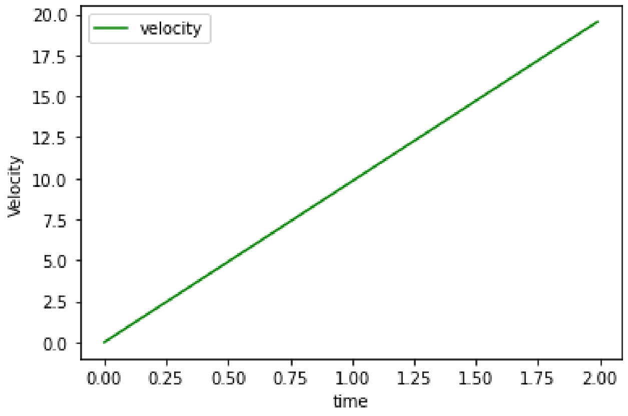
Pass the initial conditions, the time period and the integration function to the solver

Initial Conditions:

$v(0) = 0,$
$y(0) = 0,$
$a(0) = g$




Do the calculations by hand and verify if they are correct for a few timesteps.

# Simulating a particle on a 2D Plane

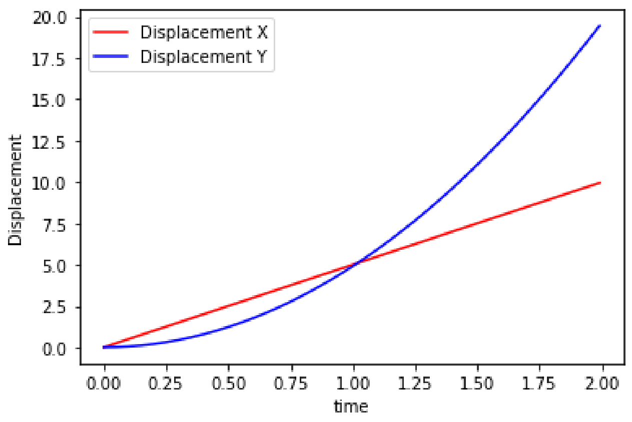
- Differential equations for each dimension:

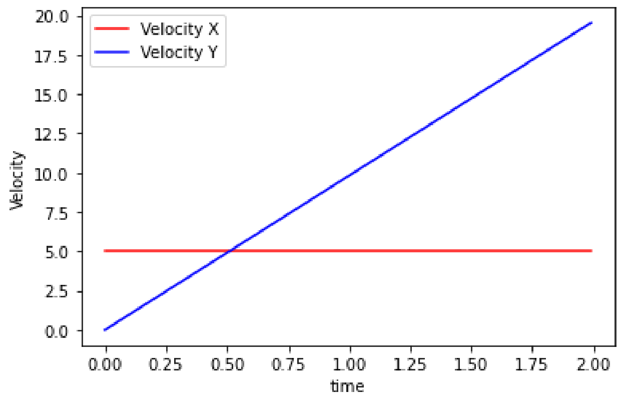
$$\dot{v_x} = 0$$

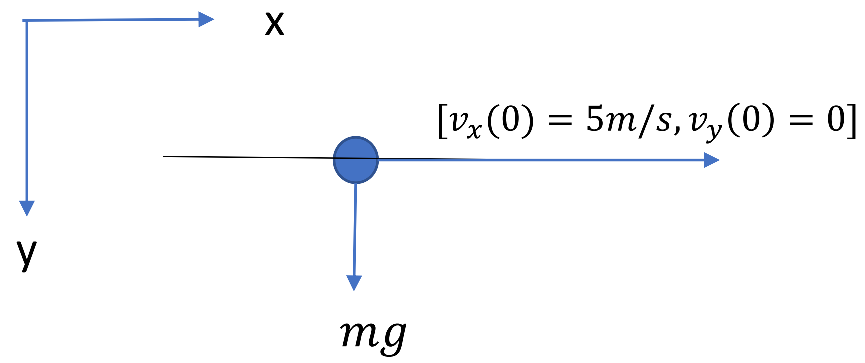
$$\dot{v_y} = g$$

$$\dot{d_x} = v_x$$

$$\dot{d_y} = v_y$$




$$Initial\ conditions: v(0) = [v_x(0), v_y(0)],$$
$$\mathrm{d}(0) = [0,0],$$
$$a(0) = [0, g]$$

# Simulating Quadrotor Dynamics

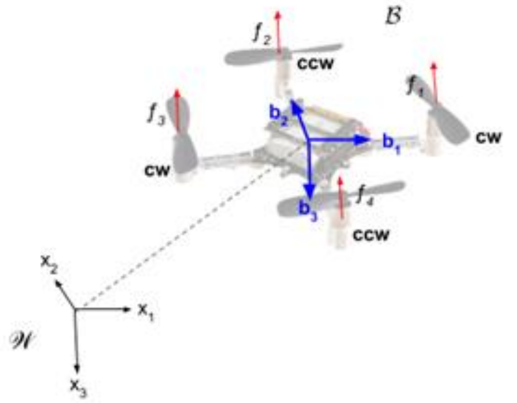



Fig: Coordinate system and thrusts generated by each rotor

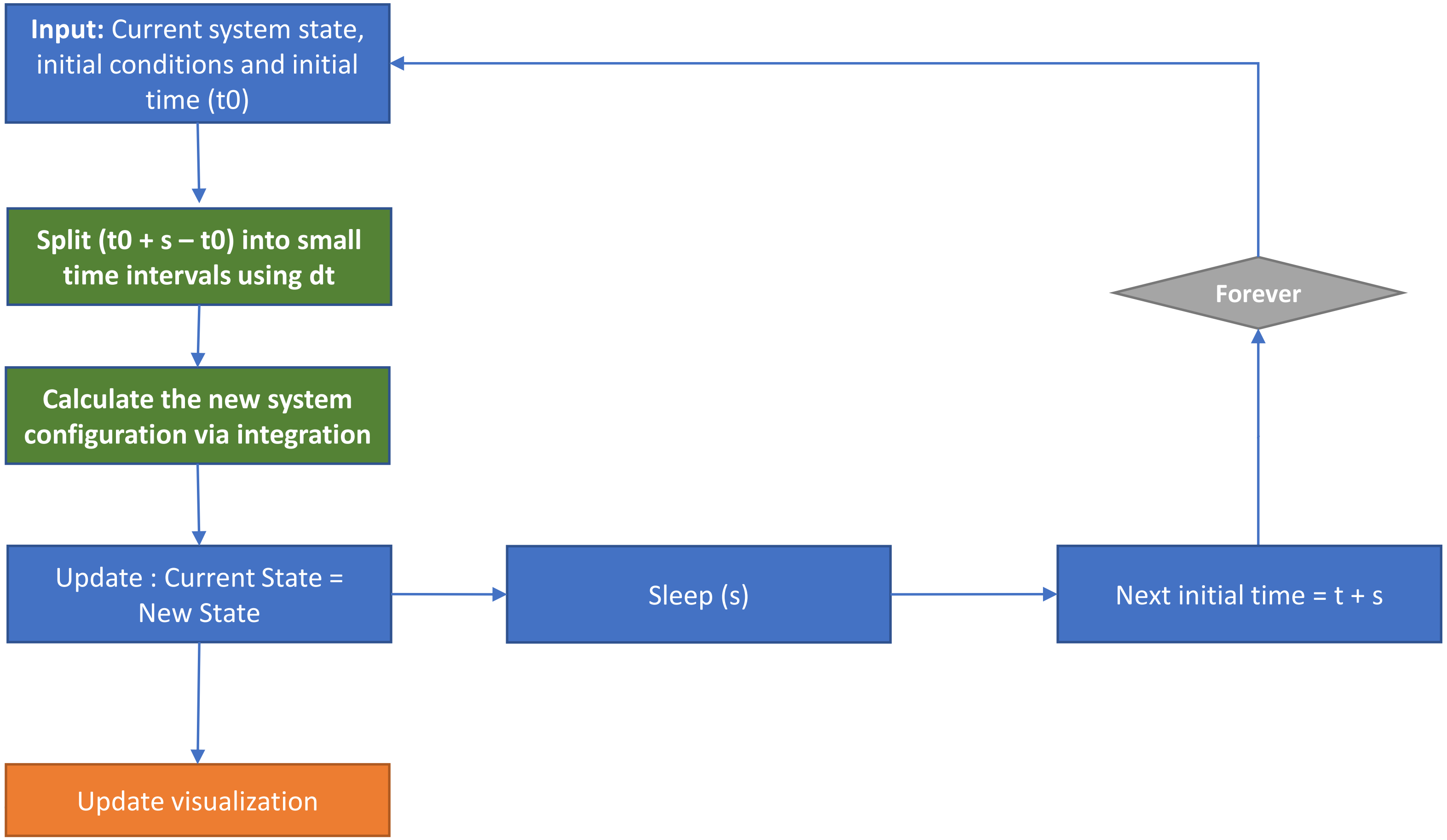
$$\dot{x} = v, \tag{2}$$
$$m\dot{v} = mge_3 - fRe_3, \tag{3}$$
$$\dot{R} = R\hat{\Omega}, \tag{4}$$
$$J\dot{\Omega} + \Omega \times J\Omega = M, \tag{5}$$

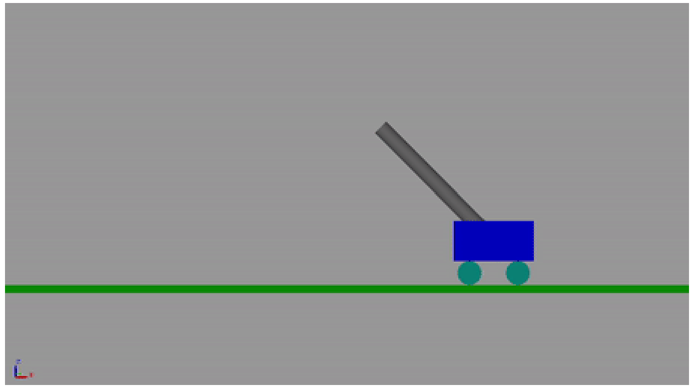
x: position vector
v: velocity vector
m: mass of the quadrotor
$e_1$,$e_2$,$e_3$: unit vectors representing each world frame axis
R: rotation matrix
Ω: angular velocity
J: Moment of inertia matrix
f: net thrust
$M_i$: Moment around *i* th body fixed axis

**Differential Equations**

**No need to memorize the equations! It’s just unfair to simulate without stating the implemented model.**

# Visualizing the Dynamics

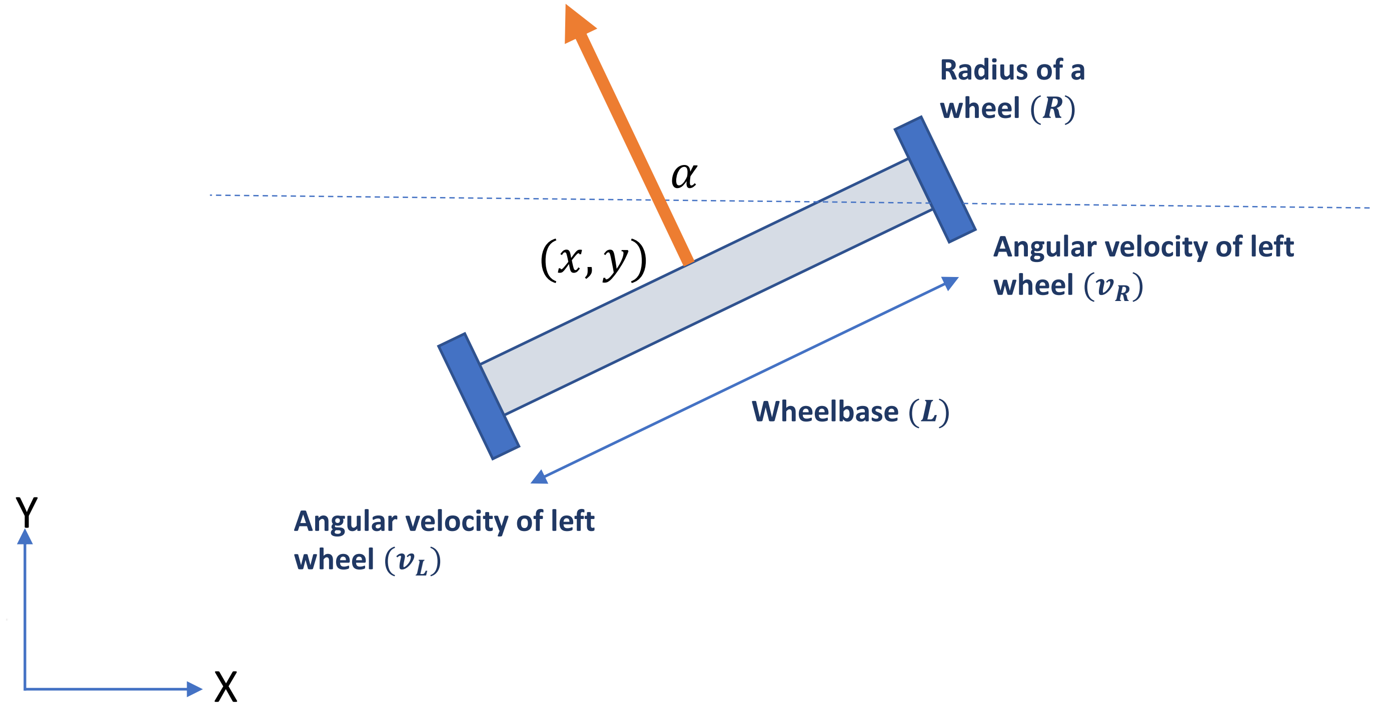


**Balancing a cartpole system**

# Differential Driven Dynamics

- **Differential dynamics**: The motion of a system actuated by two separately driven wheels on either sides. Eg.: Car, Alphabot.




$$\dot{x} = \frac{R}{2}(v_L + v_R)\cos(\alpha)$$

$$\dot{y} = \frac{R}{2}(v_L + v_R)\sin(\alpha)$$

$$\dot{\alpha} = \frac{R}{L}(v_R - v_L)$$

# Unicycle Model

- When it comes to control part in the control-feedback loop, having a lot of non-linear terms inside the controller is not intuitive.
- Unicycle model is a simplification of the differential driven dynamics that eliminate angular velocities of the wheels.
- Instead, we will control with linear velocity (V) and the heading ($\alpha$) of the robot.