上海科技大学

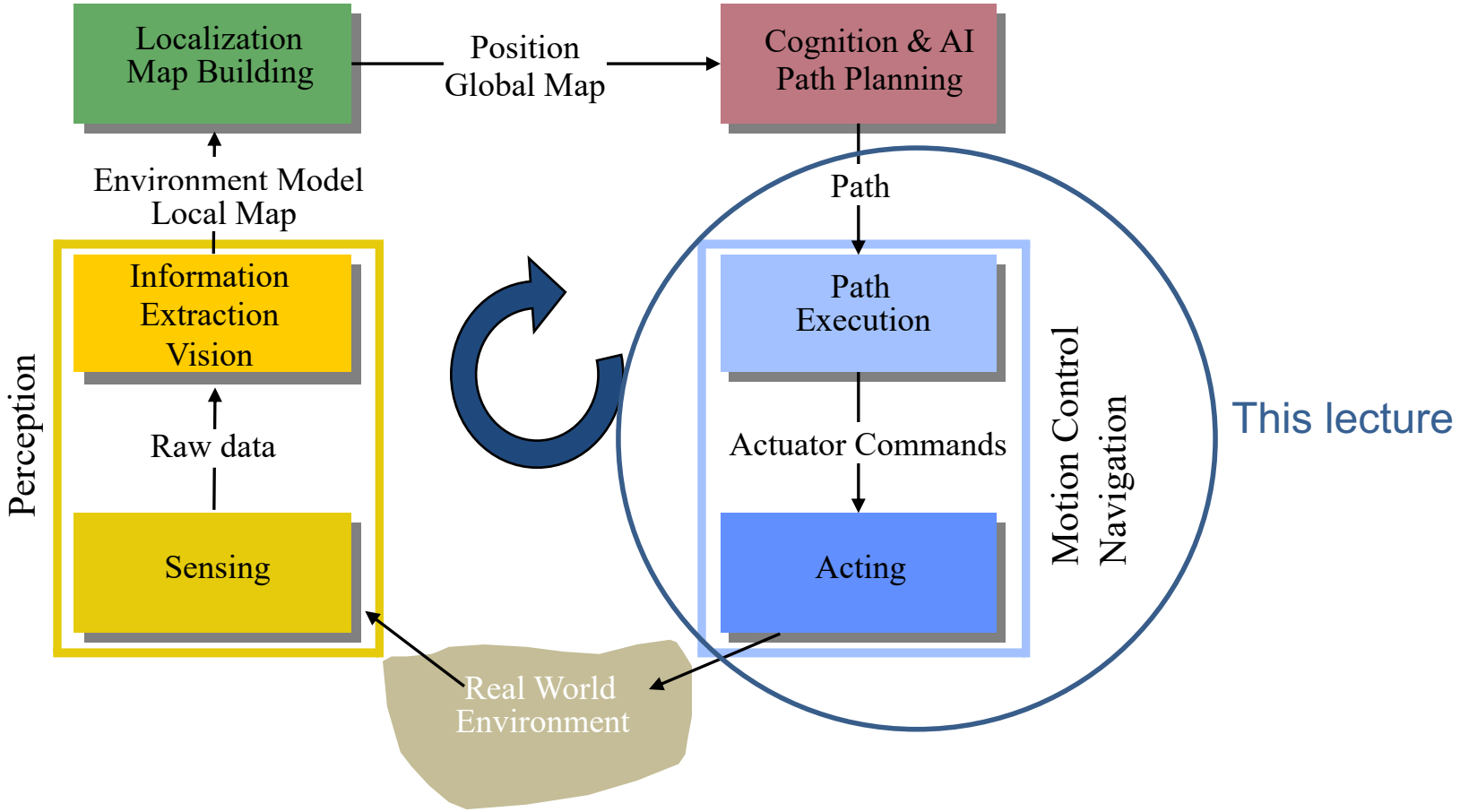
ShanghaiTech University

CS283: **Robotics Spring 2025: Control**

Sören Schwertfeger / **师泽仁**

ShanghaiTech University

General Control Scheme for Mobile Robot Systems

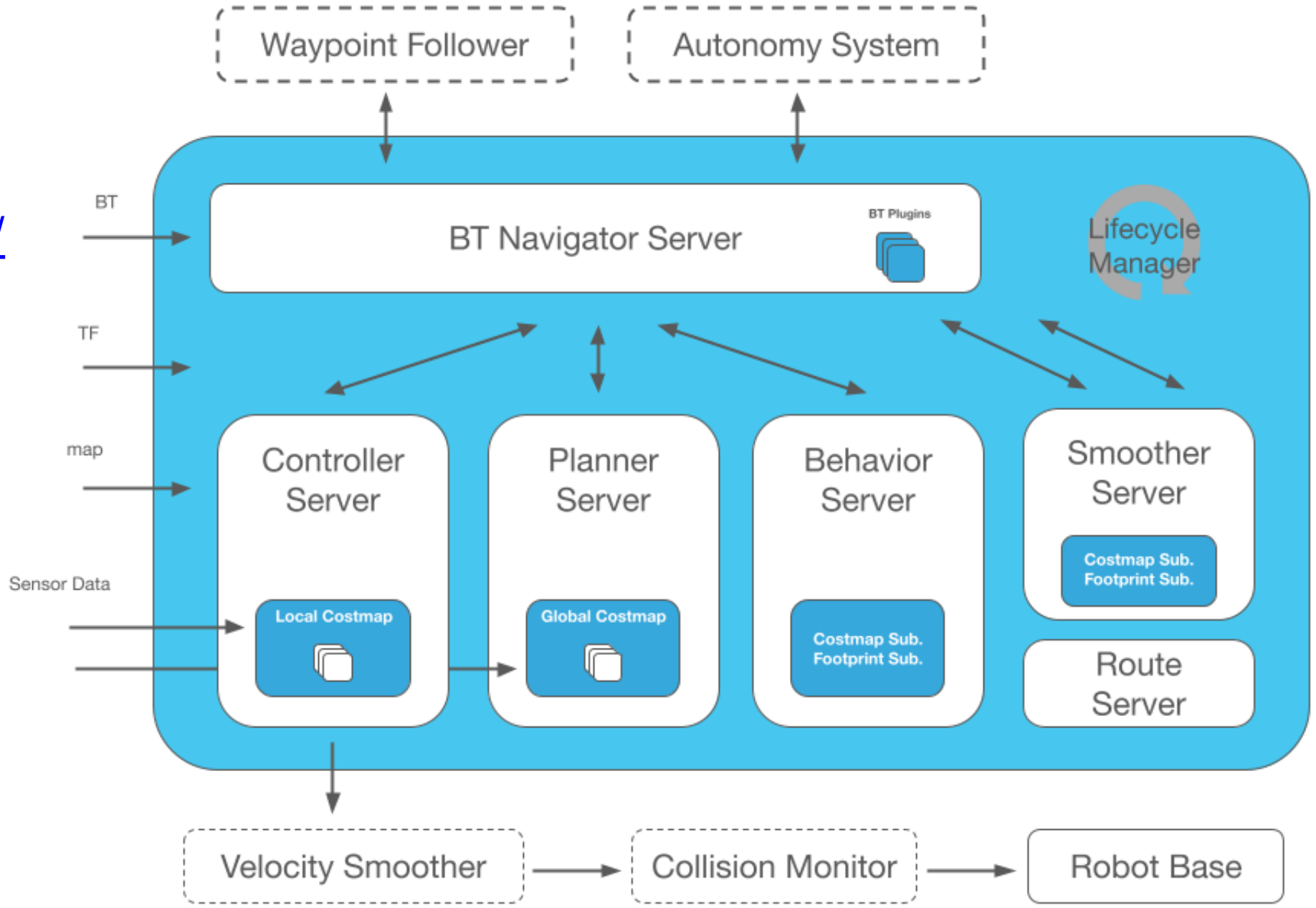


With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

ROS2 Navigation 2

navigation.ros.org/

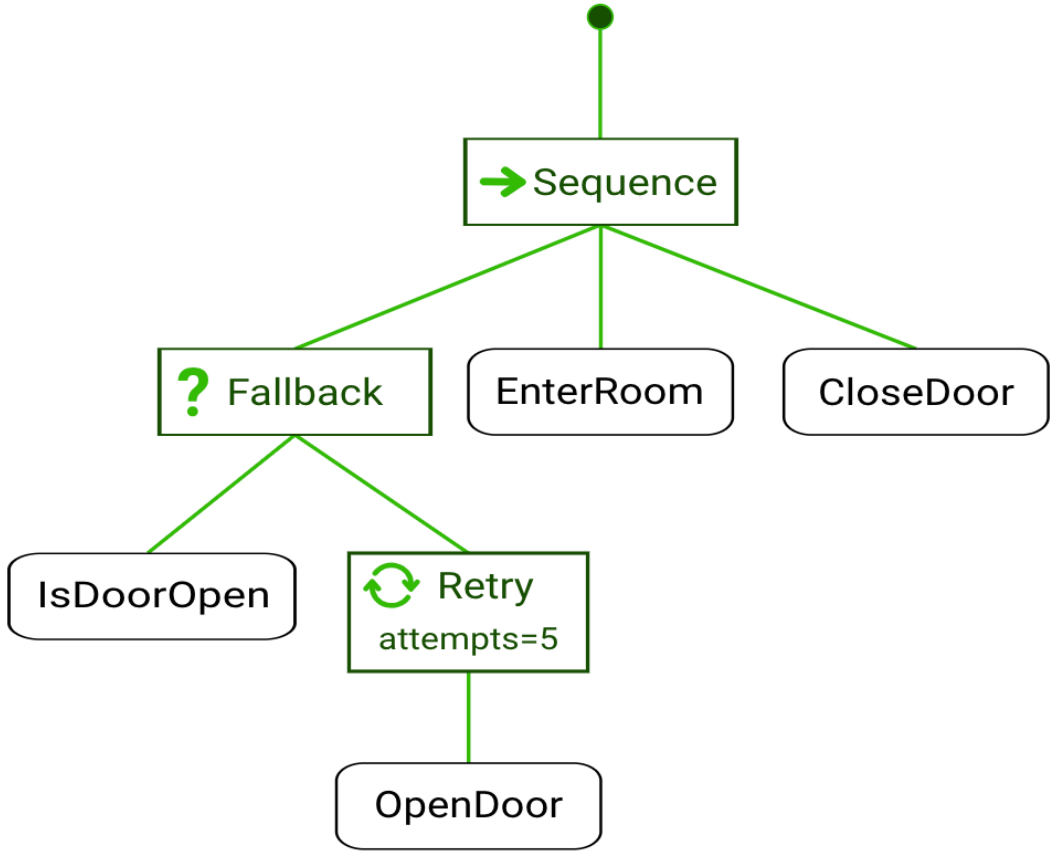
BT Navigation Server



Behavior Tree (BT)

Behavior Trees in Robotics and AI: An Introduction
Michele Colledanchise, Petter Ögren
https://arxiv.org/abs/1709.00084

- Alternative to Finite State Machine (FSM)
  - BT (supposedly) more scalable, more human-understandable and easier to reuse than FSM
  - Intrinsically hierarchical
  - Graphical representation has meaning
  - Expressive
- BehaviorTree CPP V3
  https://www.behaviortree.dev/
- Defined in XML
- Execute top down, left first (similar to DFS)



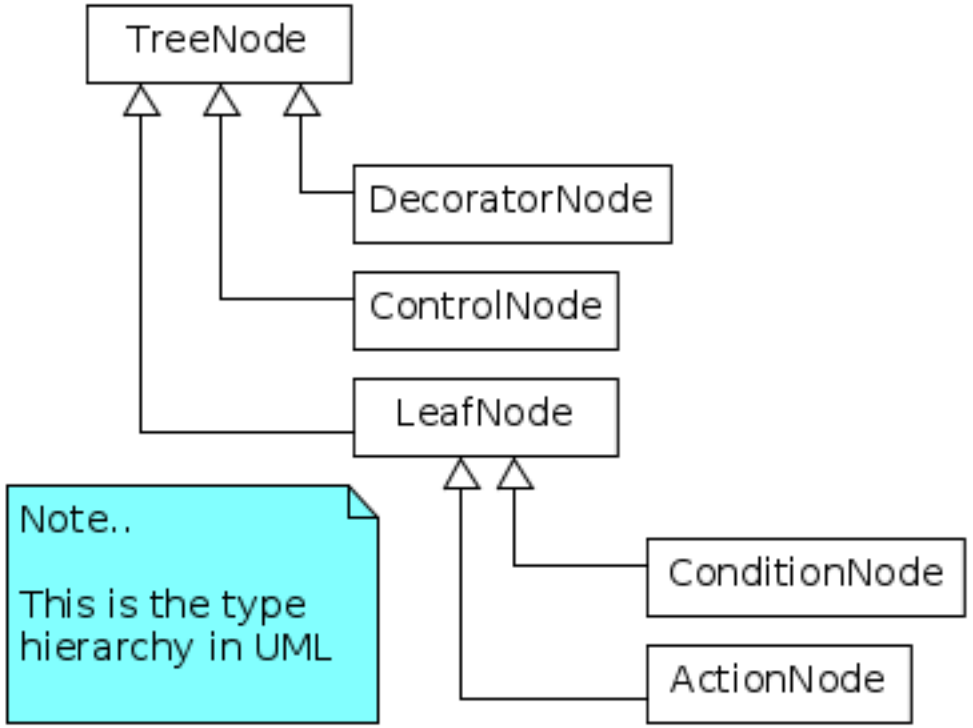
Types of BT Nodes

Control Node

Decorator Node

Action Node

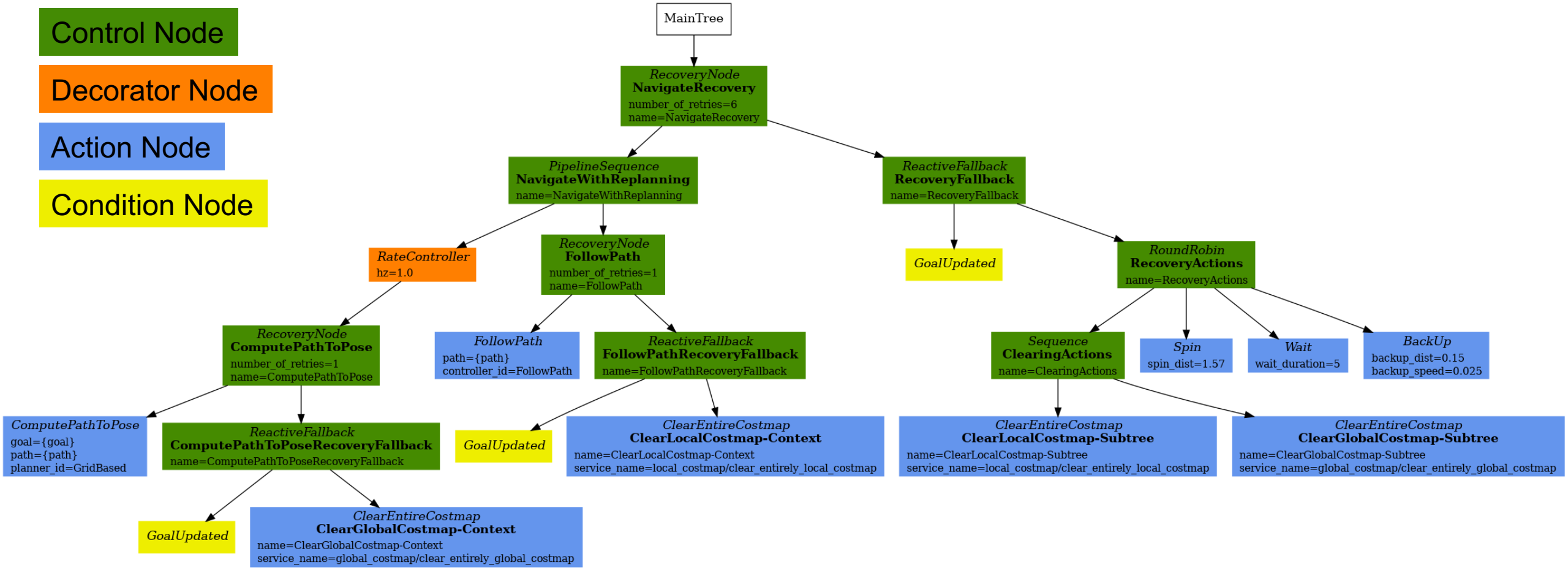
Condition Node



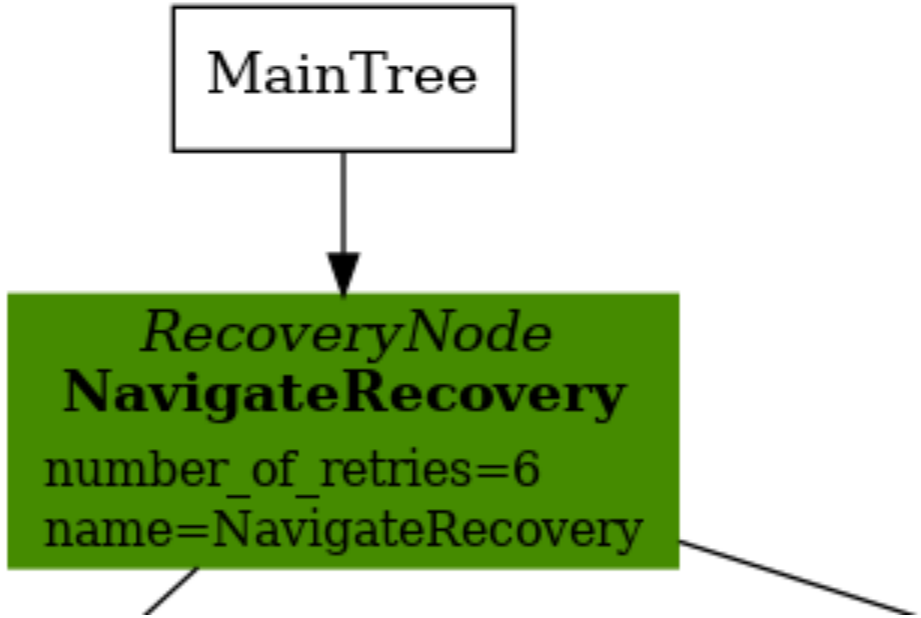
Type of TreeNode	Children Count	Notes
ControlNode	1...N	Usually, ticks a child based on the result of its siblings or/and its own state.
DecoratorNode	1	Among other things, it may alter the result of its child or tick it multiple times.
ConditionNode	0	Should not alter the system. Shall not return RUNNING.
ActionNode	0	This is the Node that "does something"

Example: Navigate To Pose With Replanning and Recovery

- Tree update rate/ tick rate: 100Hz



Recovery Node:

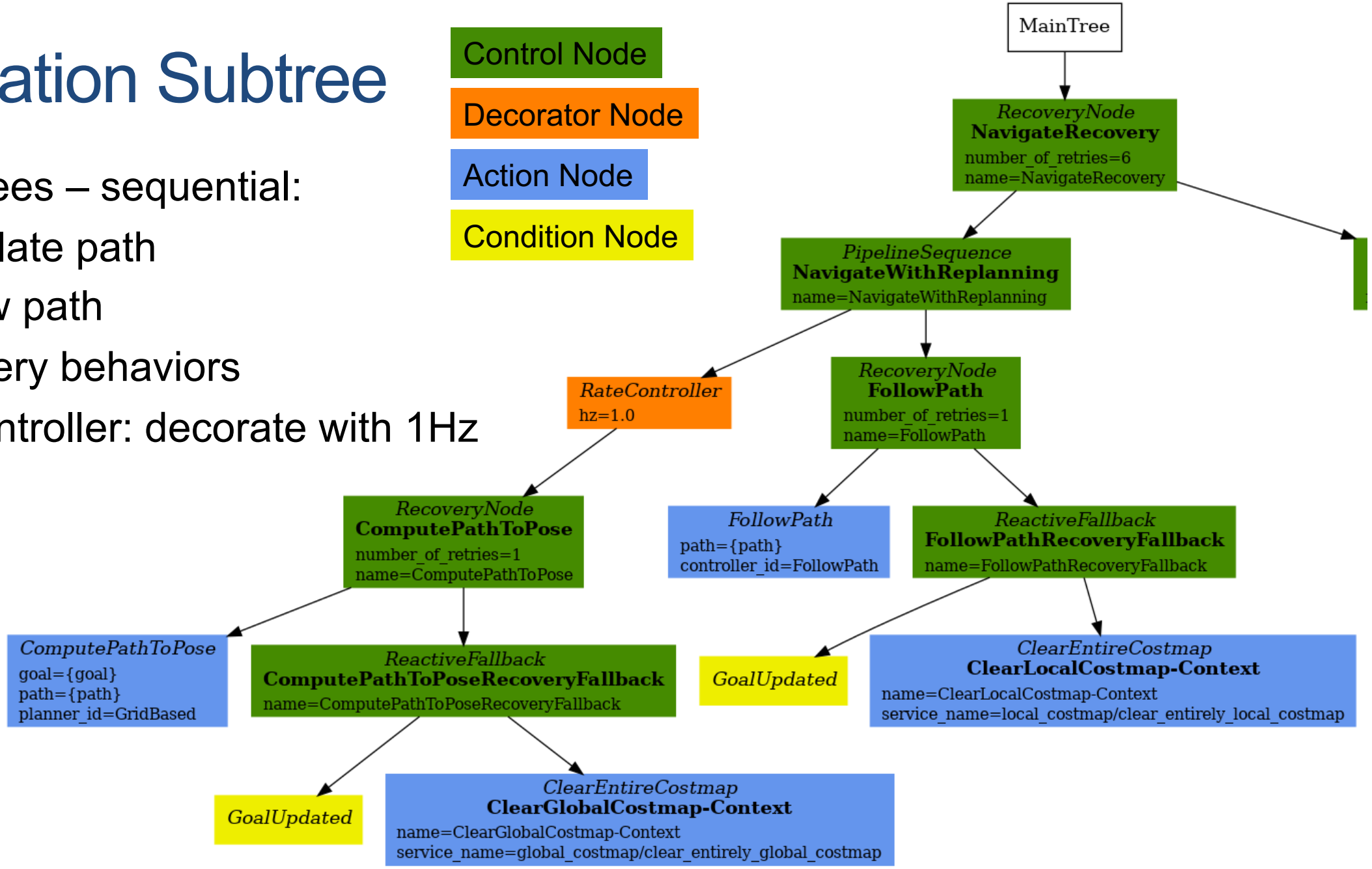


- Node must contain 2 children
- returns success if first succeeds.
- If first fails:
  - Tick the second
  - If successful retry the first
  - Repeat until first returns true or number of retires is up

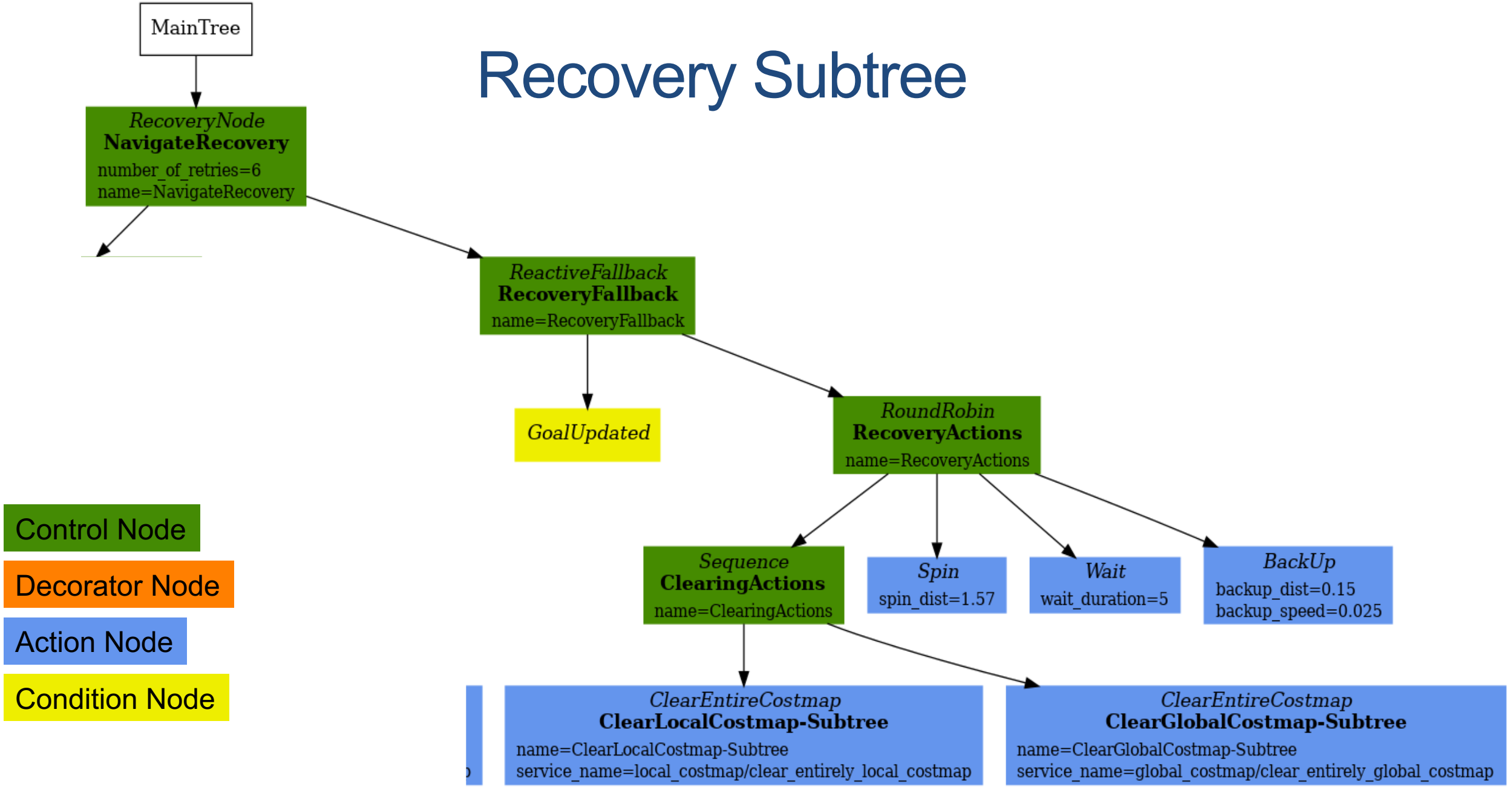
- Children of Navigate Recovery Node:
  - Navigation Subtree
  - Recovery Subtree

Navigation Subtree

- 2 Subtrees – sequential:
  - Calculate path
  - Follow path
- + recovery behaviors
- RateController: decorate with 1Hz

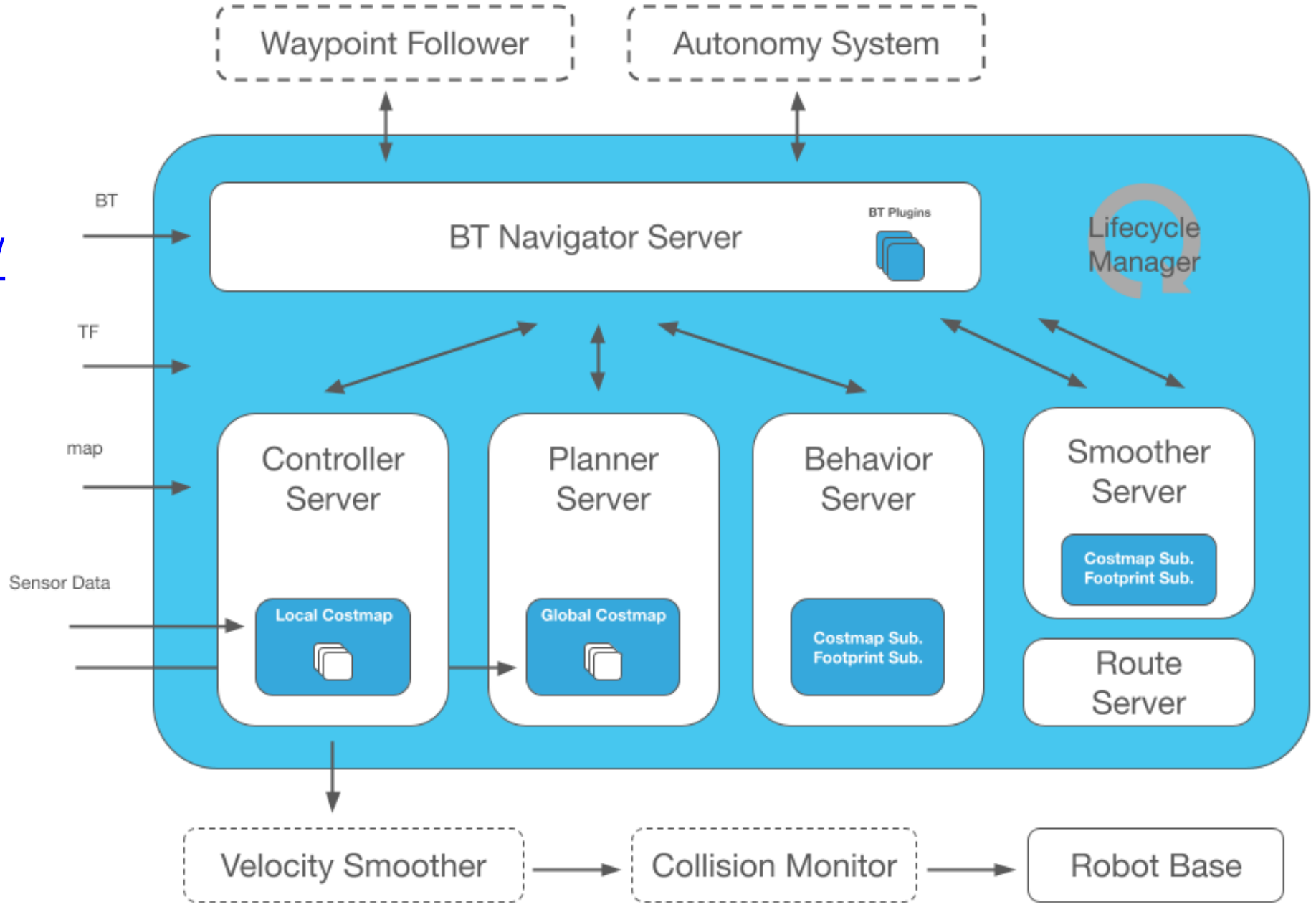


Recovery Subtree

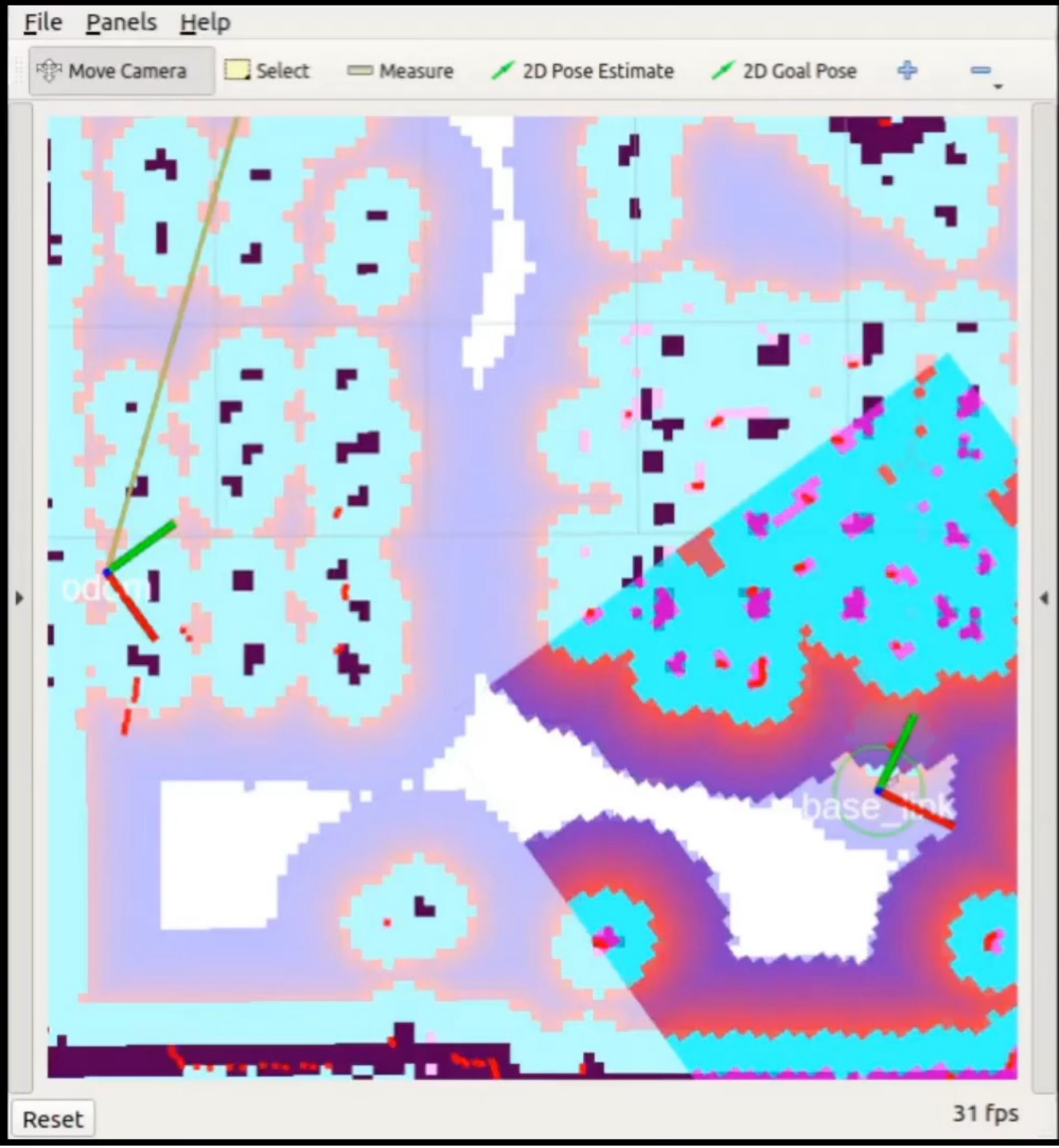


ROS2 Navigation 2

navigation.ros.org/



Local Costmap



https://www.youtube.com/watch?v=6oopJWjf6do

Navigation Plugins

Behavior-Tree Navigators

Plugin Name	Creator	Description
NavigateToPoseNavigator	Steve Macenski	Point-to-point navigation via a behavior tree action server
NavigateThroughPosesNavigator	Steve Macenski	Point-through-points navigation via a behavior tree action server
CoverageNavigator	Steve Macenski	Complete coverage navigation (Cartesian or GPS) via a BTs

https://navigation.ros.org/plugins/index.html

Costmap Layers

Plugin Name	Creator	Description
Voxel Layer	Eitan Marder-Eppstein	Maintains persistant 3D voxel layer using depth and laser sensor readings and raycasting to clear free space
Range Layer	David Lu	Uses a probabalistic model to put data from sensors that publish range msgs on the costmap
Static Layer	Eitan Marder-Eppstein	Gets static `map` and loads occupancy information into costmap
Inflation Layer	Eitan Marder-Eppstein	Inflates lethal obstacles in costmap with exponential decay
Obstacle Layer	Eitan Marder-Eppstein	Maintains persistent 2D costmap from 2D laser scans with raycasting to clear free space
Spatio-Temporal Voxel Layer	Steve Macenski	Maintains temporal 3D sparse volumetric voxel grid with decay through sensor models
Non-Persistent Voxel Layer	Steve Macenski	Maintains 3D occupancy grid consisting only of the most sets of measurements
Denoise Layer	Andrey Ryzhikov	Filters noise-induced standalone obstacles or small obstacles groups

Costmap Filters

Plugin Name	Creator	Description
Keepout Filter	Alexey Merzlyakov	Maintains keep-out/safety zones and preferred lanes for moving
Speed Filter	Alexey Merzlyakov	Limits maximum velocity of robot in speed restriction areas
Binary Filter	Alexey Merzlyakov	Enables binary (boolean) mask behavior to trigger actions.

Controllers

Plugin Name	Creator	Description	Drivetrain support
DWB Controller	David Lu!!	A highly configurable DWA implementation with plugin interfaces	Differential, Omnidirectional, Legged
TEB Controller	Christoph Rösmann	A MPC-like controller suitable for ackermann, differential, and holonomic robots.	**Ackermann**, Legged, Omnidirectional, Differential
Regulated Pure Pursuit	Steve Macenski	A service / industrial robot variation on the pure pursuit algorithm with adaptive features.	**Ackermann**, Legged, Differential
MPPI Controller	Steve Macenski Aleksei Budyakov	A predictive MPC controller with modular & custom cost functions that can accomplish many tasks.	Differential, Omni, **Ackermann**
Rotation Shim Controller	Steve Macenski	A “shim” controller to rotate to path heading before passing to main controller for tracking.	Differential, Omni, model rotate in place
Graceful Controller	Alberto Tudela	A controller based on a pose-following control law to generate smooth trajectories.	Differential

Planners

Plugin Name	Creator	Description	Drivetrain support
NavFn Planner	Eitan Marder-Eppstein & Kurt Konolige	A navigation function using A* or Dijkstras expansion, assumes 2D holonomic particle	Differential, Omnidirectional, Legged
SmacPlannerHybrid (formerly *SmacPlanner*)	Steve Macenski	A SE2 Hybrid-A* implementation using either Dubin or Reeds-shepp motion models with smoother and multi-resolution query. Cars, car-like, and ackermann vehicles. Kinematically feasible.	**Ackermann**, Differential, Omnidirectional, Legged
SmacPlanner2D	Steve Macenski	A 2D A* implementation Using either 4 or 8 connected neighborhoods with smoother and multi-resolution query	Differential, Omnidirectional, Legged
SmacPlannerLattice	Steve Macenski	An implementation of State Lattice Planner using pre-generated minimum control sets for kinematically feasible planning with any type of vehicle imaginable. Includes generator script for Ackermann, diff, omni, and legged robots.	Differential, Omnidirectional, Ackermann, Legged, Arbitrary / Custom
ThetaStarPlanner	Anshumaan Singh	An implementaion of Theta* using either 4 or 8 connected neighborhoods, assumes the robot as a 2D holonomic particle	Differential, Omnidirectional

Smoothers

Plugin Name	Creator	Description
Simple Smoother	Steve Macenski	A simple path smoother for infeasible (e.g. 2D) planners
Constrained Smoother	Matej Vargovcik & Steve Macenski	A path smoother using a constraints problem solver to optimize various criteria such as smoothness or distance from obstacles, maintaining minimum turning radius
Savitzky-Golay Smoother	Steve Macenski	A path smoother using a Savitzky-Golay filter to smooth the path via digital signal processing to remove noise from the path.

Behaviors

Plugin Name	Creator	Description
Clear Costmap	Eitan Marder-Eppstein	A service to clear the given costmap in case of incorrect perception or robot is stuck
Spin	Steve Macenski	Rotate behavior of configurable angles to clear out free space and nudge robot out of potential local failures
Back Up	Brian Wilcox	Back up behavior of configurable distance to back out of a situation where the robot is stuck
Wait	Steve Macenski	Wait behavior with configurable time to wait in case of time based obstacle like human traffic or getting more sensor data
Drive On Heading	Joshua Wallace	Drive on heading behavior with configurable distance to drive
Assisted Teleop	Joshua Wallace	AssistedTeleop behavior that scales teleop commands to prevent collisions.

Waypoint Task Executors

Plugin Name	Creator	Description
WaitAtWaypoint	Fetullah Atas	A plugin to execute a wait behavior on waypoint arrivals.
PhotoAtWaypoint	Fetullah Atas	A plugin to take and save photos to specified directory on waypoint arrivals.
InputAtWaypoint	Steve Macenski	A plugin to wait for user input before moving onto the next waypoint.

Goal Checkers

Plugin Name	Creator	Description
SimpleGoalChecker	David Lu!!	A plugin check whether robot is within translational distance and rotational distance of goal.
StoppedGoalChecker	David Lu!!	A plugin check whether robot is within translational distance , rotational distance of goal, and velocity threshold.

Progress Checkers

Plugin Name	Creator	Description
SimpleProgressChecker	David Lu!!	A plugin to check whether the robot was able to move a minimum distance in a given time to make progress towards a goal
PoseProgressChecker	Guillaume Doisy	A plugin to check whether the robot was able to move a minimum distance or angle in a given time to make progress towards a goal

Behavior Tree Nodes

Action Plugin Name	Creator	Description
Back Up Action	Michael Jeronimo	Calls backup behavior action
Drive On Heading Action	Joshua Wallace	Calls drive on heading behavior action
Assisted Teleop Action	Joshua Wallace	Calls assisted teleop behavior action
Clear Entire Costmap Service	Carl Delsey	Calls clear entire costmap service
Clear Costmap Except Region Service	Guillaume Doisy	Calls clear costmap except region service
Clear Costmap Around Robot Service	Guillaume Doisy	Calls clear costmap around robot service
Compute Path to Pose Action	Michael Jeronimo	Calls Nav2 planner server
Smooth Path Action	Matej Vargovcik	Calls Nav2 smoother server
Follow Path Action	Michael Jeronimo	Calls Nav2 controller server
Navigate to Pose Action	Michael Jeronimo	BT Node for other BehaviorTree.CPP BTs to call Navigation2 as a subtree action
Reinitalize Global Localization Service	Carl Delsey	Reinitialize AMCL to a new pose
Spin Action	Carl Delsey	Calls spin behavior action
Wait Action	Steve Macenski	Calls wait behavior action
Truncate Path	Francisco Martín	Modifies a path making it shorter
Truncate Path Local	Matej Vargovcik	Extracts a path section around robot
Planner Selector	Pablo Iñigo Blasco	Selects the global planner based on a topic input, otherwises uses a default planner id
Controller Selector	Pablo Iñigo Blasco	Selects the controller based on a topic input, otherwises uses a default controller id
Goal Checker Selector	Pablo Iñigo Blasco	Selects the goal checker based on a topic input, otherwises uses a default goal checker id
Smoother Selector	Owen Hooper	Selects the smoother based on a topic input, otherwises uses a default smoother id
Progress Checker Selector	Steve Macenski	Selects the progress checker based on a topic input, otherwises uses a default progress checker id
Navigate Through Poses	Steve Macenski	BT Node for other BehaviorTree.CPP BTs to call Nav2's NavThroughPoses action
Remove Passed Goals	Steve Macenski	Removes goal poses passed or within a tolerance for culling old viapoints from path re-planning
Compute Path Through Poses	Steve Macenski	Computes a path through a set of poses rather than a single end goal pose using the planner plugin specified
Cancel Control Action	Pradheep Padmanabhan	Cancels Nav2 controller server
Cancel BackUp Action	Pradheep Padmanabhan	Cancels backup behavior action
Cancel Spin Action	Pradheep Padmanabhan	Cancels spin behavior action
Cancel Wait Action	Pradheep Padmanabhan	Cancels wait behavior action
Cancel Drive on Heading Action	Joshua Wallace	Cancels drive on heading behavior action
Cancel Assisted Teleop Action	Joshua Wallace	Cancels assisted teleop behavior action
Cancel Complete Coverage Action	Steve Macenski	Cancels compute complete coverage
Compute Complete Coverage Path Action	Steve Macenski	Calls coverage planner server

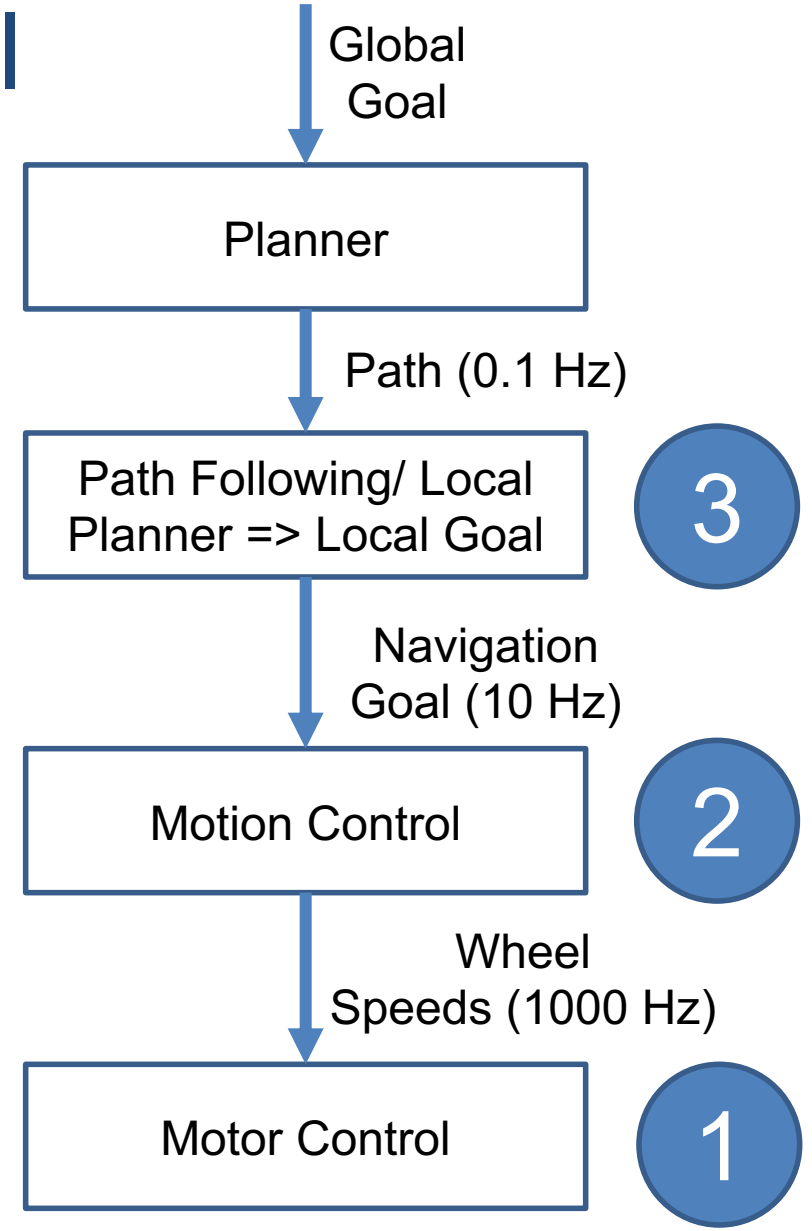
Condition Plugin Name	Creator	Description
Goal Reached Condition	Carl Delsey	Checks if goal is reached within tol.
Goal Updated Condition	Aitor Miguel Blanco	Checks if goal is preempted.
Globally Updated Goal Condition	Joshua Wallace	Checks if goal is preempted in the global BT context
Initial Pose received Condition	Carl Delsey	Checks if initial pose has been set
Is Stuck Condition	Michael Jeronimo	Checks if robot is making progress or stuck
Transform Available Condition	Steve Macenski	Checks if a TF transformation is available. When succeeds returns success for subsequent calls.
Distance Traveled Condition	Sarthak Mittal	Checks is robot has traveled a given distance.
Time Expired Condition	Sarthak Mittal	Checks if a given time period has passed.
Is Battery Low Condition	Sarthak Mittal	Checks if battery percentage is below a specified value.
Is Path Valid Condition	Joshua Wallace	Checks if a path is valid by making sure there are no LETHAL obstacles along the path.
Path Expiring Timer	Joshua Wallace	Checks if the timer has expired. The timer is reset if the path gets updated.
Are Error Codes Present	Joshua Wallace	Checks if the specified error codes are present.
Would A Controller Recovery Help	Joshua Wallace	Checks if a controller recovery could help clear the controller server error code.
Would A Planner Recovery Help	Joshua Wallace	Checks if a planner recovery could help clear the planner server error code.
Would A Smoother Recovery Help	Joshua Wallace	Checks if a Smoother recovery could help clear the smoother server error code.
Is Battery Charging Condition	Alberto Tudela	Checks if the battery is charging.

Decorator Plugin Name	Creator	Description
Rate Controller	Michael Jeronimo	Throttles child node to a given rate
Distance Controller	Sarthak Mittal	Ticks child node based on the distance traveled by the robot
Speed Controller	Sarthak Mittal	Throttles child node to a rate based on current robot speed.
Goal Updater	Francisco Martín	Updates the goal received via topic subscription.
Single Trigger	Steve Macenski	Triggers nodes/subtrees below only a single time per BT run.
PathLongerOnApproach	Pradheep Padmanabhan	Triggers child nodes if the new global path is significantly larger than the old global path on approach to the goal

Control Plugin Name	Creator	Description
Pipeline Sequence	Carl Delsey	A variant of a sequence node that will re-tick previous children even if another child is running
Recovery	Carl Delsey	Node must contain 2 children and returns success if first succeeds. If first fails, the second will be ticked. If successful, it will retry the first and then return its value
Round Robin	Mohammad Haghighipanah	Will tick `i` th child until a result and move on to `i+1`

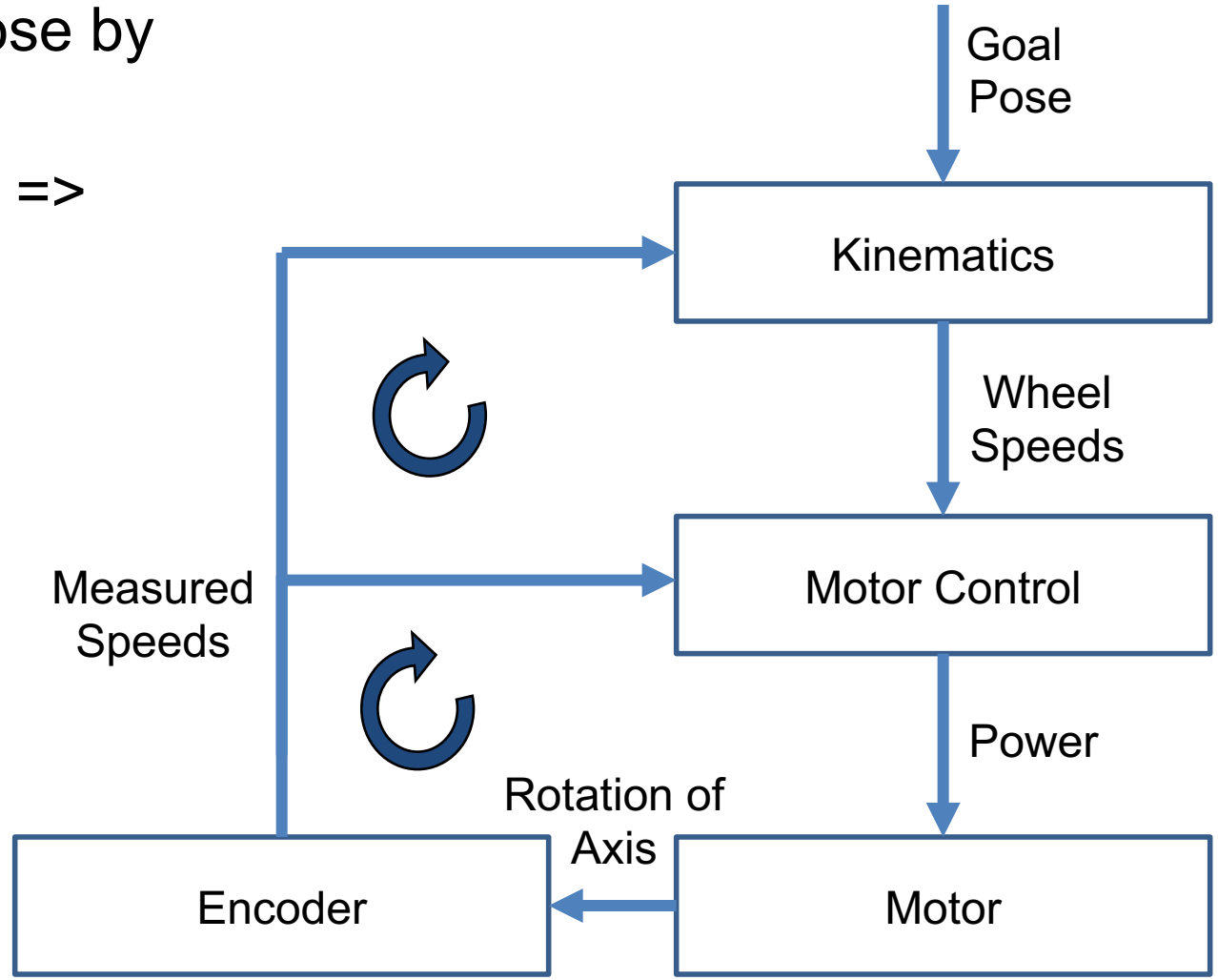
Navigation, Motion & Motor Control

- Navigation/ Motion Control:
  - Where to drive to **next** in order to reach goal
  - Output: motion vector (direction) and speed
  - For example:
    - follow path (Big Model)
    - go to unexplored area (Big Model)
    - drive forward (Small Model)
    - be attracted to goal area (Small Model)
- Motion Control:
  - How use propulsion to achieve motion vector
- Motor Control:
  - How much power to achieve propulsion (wheel speed)



Overview

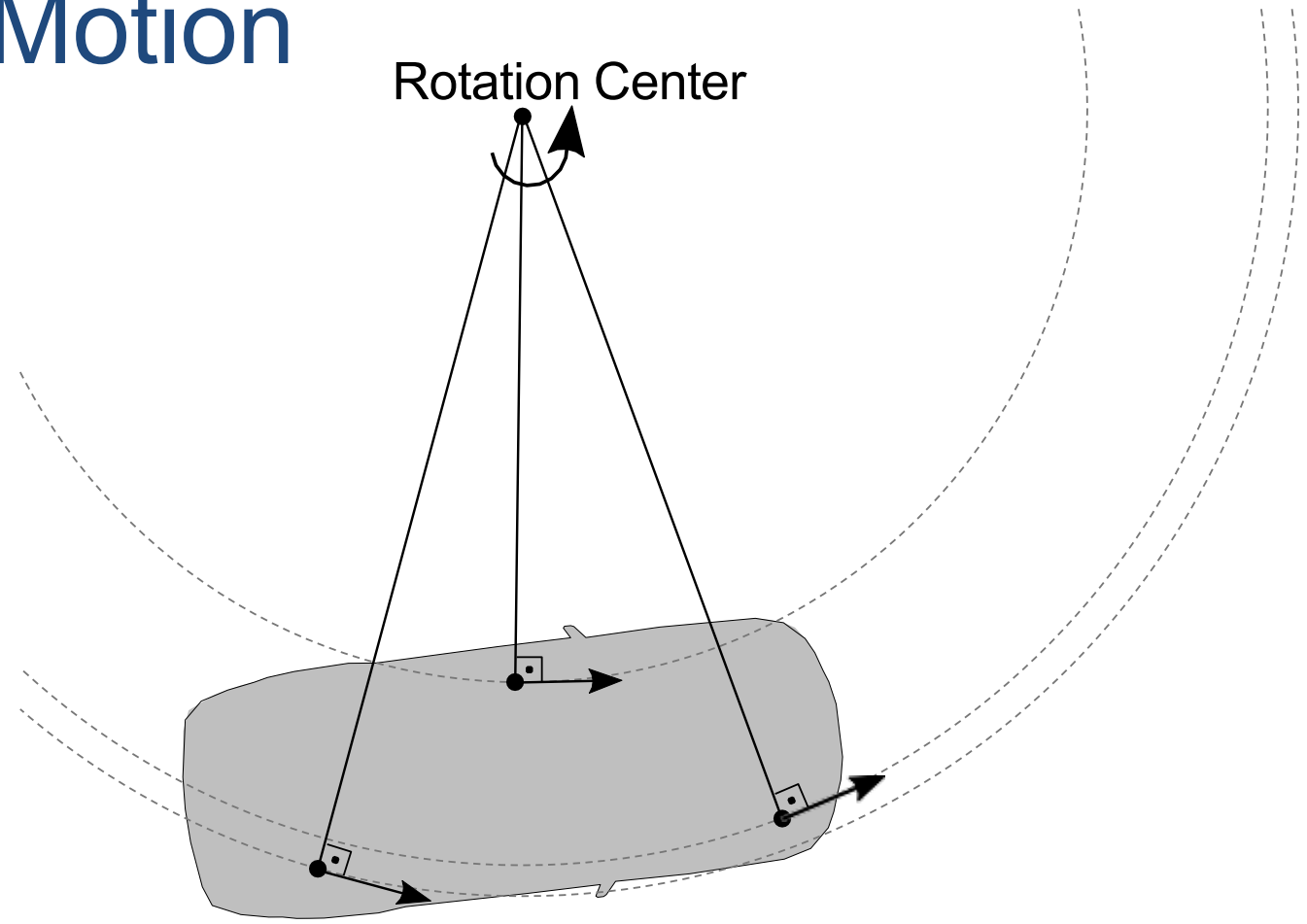
- Assume we have a pose - close by from local planner
- Calculate Inverse Kinematics =>
- Desired wheel speeds
  - Typically, not just one wheel =>
  - Many motor controllers, motors, encoders
- Motor control loop
- Pose control loop



MOTION CONTROL

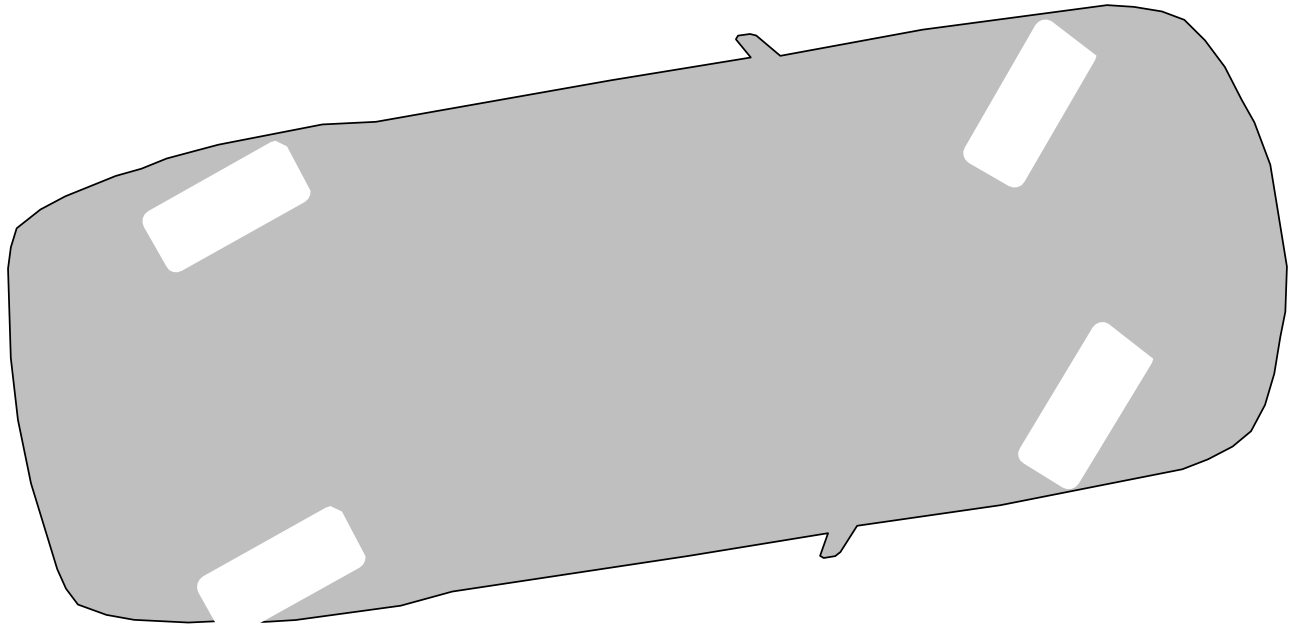
Kinematic Model

Rigid Body Motion



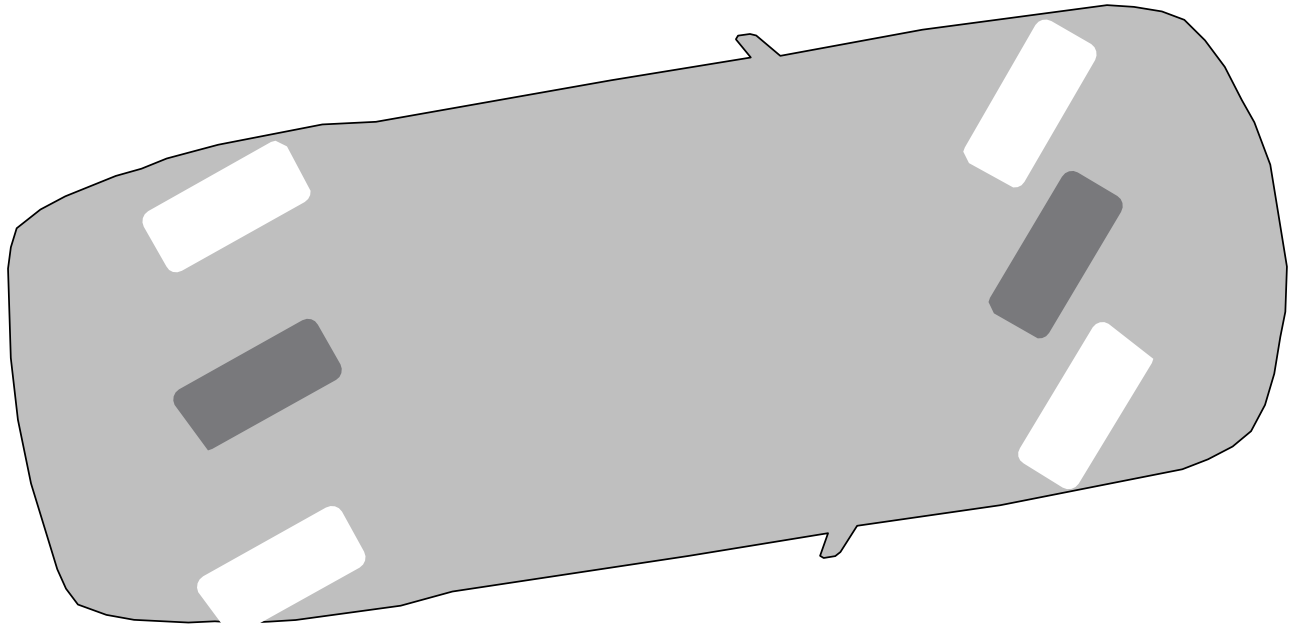
- Different points on the rigid body move along different circular trajectories

Kinematic Bicycle Model



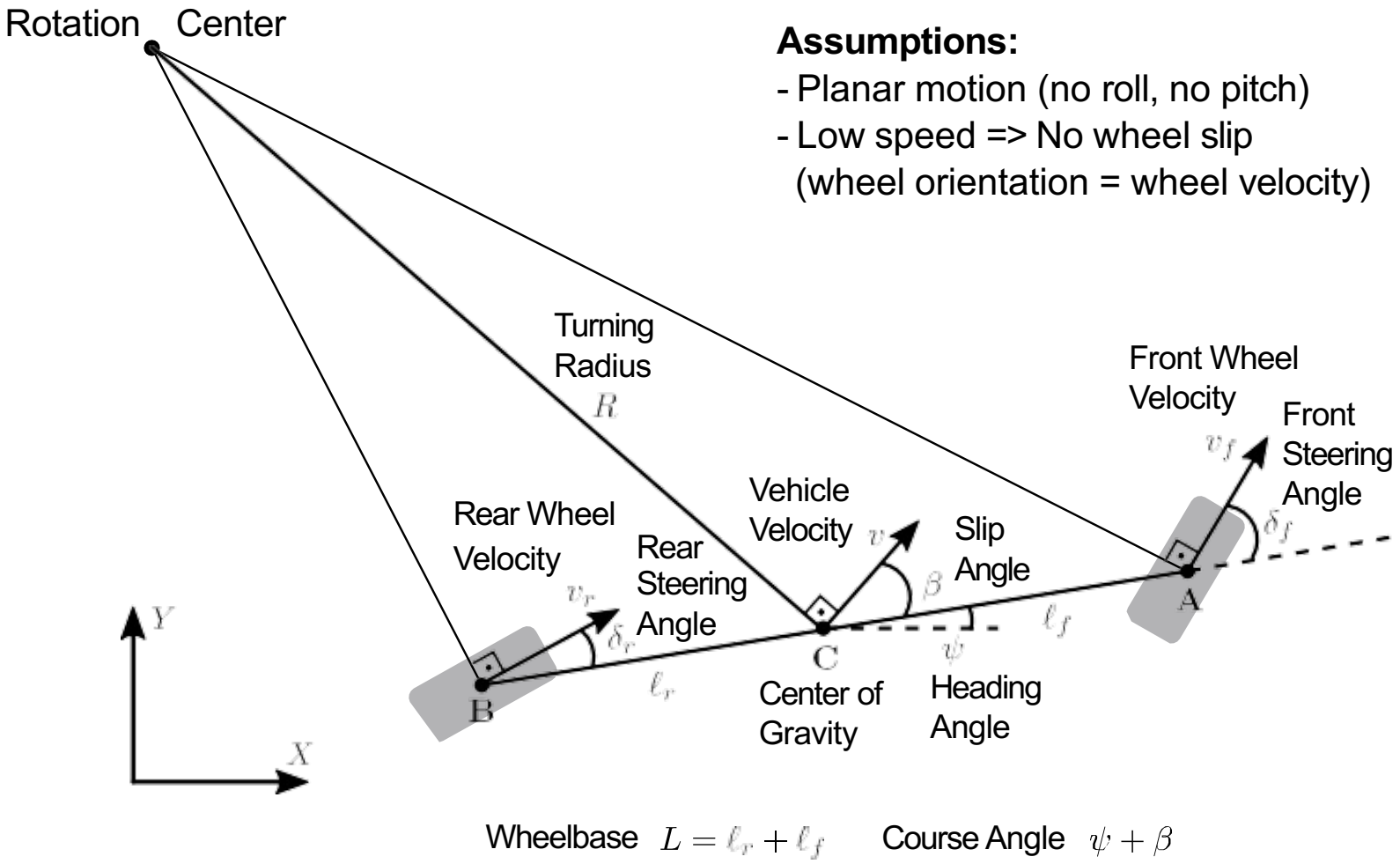
- The **kinematic bicycle model** approximates the 4 wheels with 2 imaginary wheels

Kinematic Bicycle Model



- The **kinematic bicycle model** approximates the 4 wheels with 2 imaginary wheels

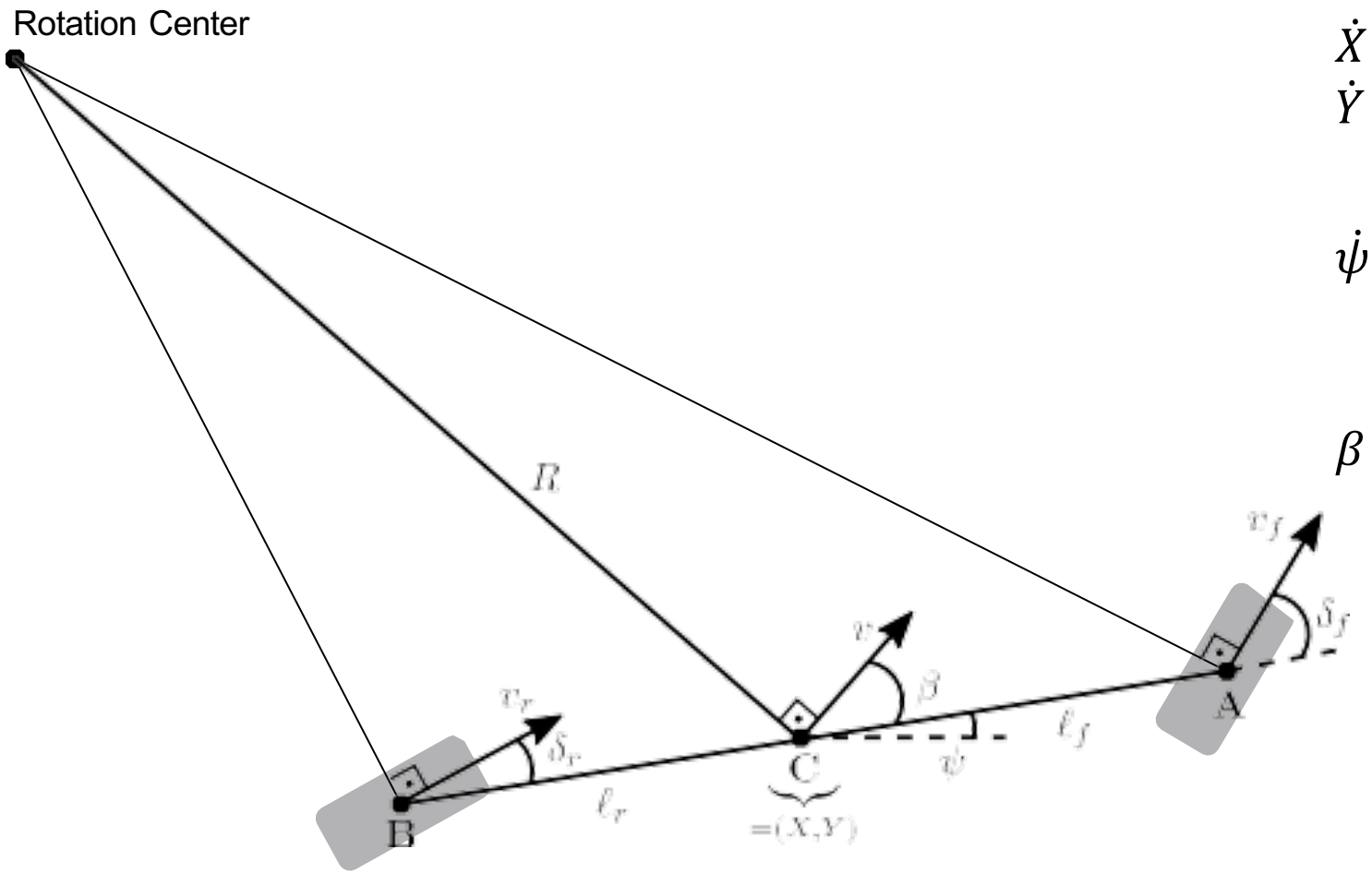
Kinematic Bicycle Model



- The **kinematic bicycle model** approximates the 4 wheels with 2 imaginary wheels

Kinematic Bicycle Model

Model



Motion Equations

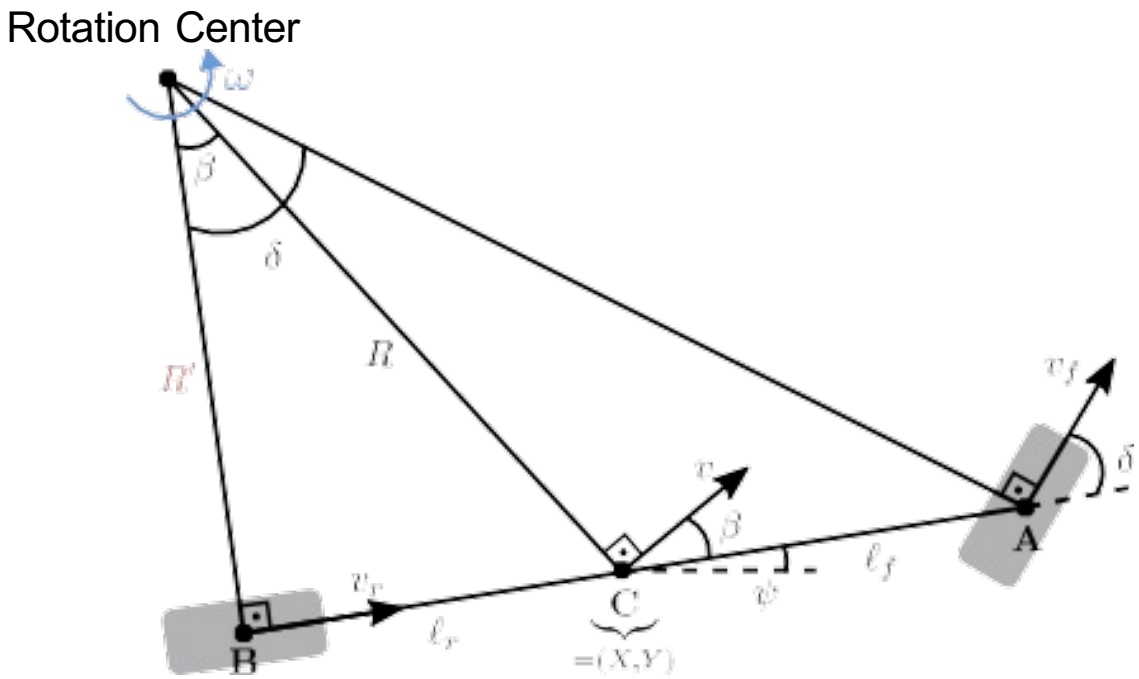
$$\dot{X} = v\cos(\psi + \beta)$$
$$\dot{Y} = v\sin(\psi + \beta)$$

$$\dot{\psi} = \frac{v\cos(\beta)}{\ell_f + \ell_r}\left(\tan(\delta_f) - \tan(\delta_r)\right)$$

$$\beta = \tan^{-1}\left(\frac{\ell_f \tan(\delta_r) + \ell_r \tan(\delta_f)}{\ell_f + \ell_r}\right)$$

Kinematic Bicycle Model Front Steering Only

Model



Motion Equations

$$\dot{X} = v\cos(\psi + \beta)$$
$$\dot{Y} = v\sin(\psi + \beta)$$

$$\dot{\psi} = \frac{v\cos(\beta)}{\ell_f + \ell_r}\tan(\delta)$$

$$\beta = \tan^{-1}\left(\frac{\ell_r \tan(\delta)}{\ell_f + \ell_r}\right)$$

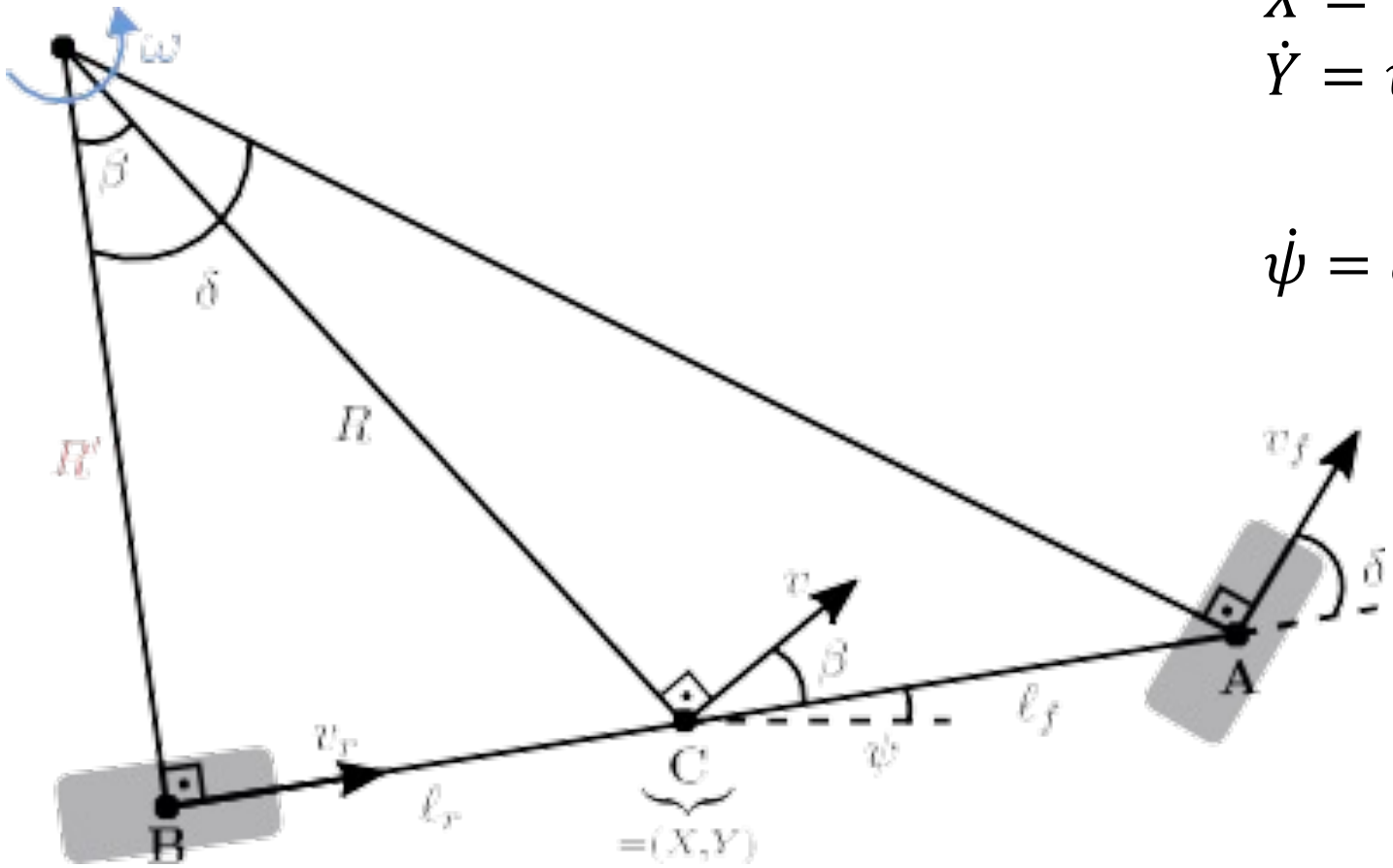
$$\tan\delta = \frac{\ell_f + \ell_r}{R'} \quad \Rightarrow \quad \frac{1}{R'} = \frac{\tan\delta}{\ell_f + \ell_r} \quad \Rightarrow \quad \tan\beta = \frac{\ell_r}{R'} = \frac{\ell_r \tan\delta}{\ell_f + \ell_r}$$

$$\cos\beta = \frac{R'}{R} \quad \Rightarrow \quad \frac{1}{R} = \frac{\cos\beta}{R'} \quad \Rightarrow \quad \dot{\psi} = \omega = \frac{v}{R} = \frac{v\cos\beta}{R'} = \frac{v\cos\beta}{\ell_f + \ell\_r}\tan\delta$$

Kinematic Bicycle Model

Model

Rotation Center

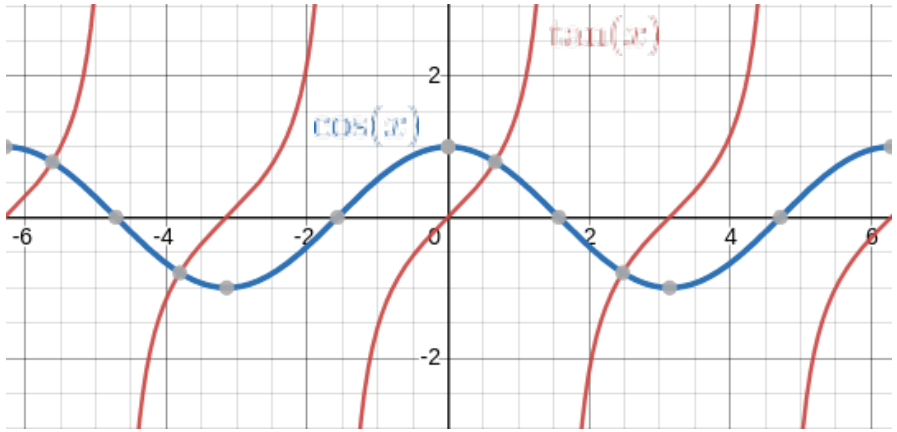


Motion Equations

$$\dot{X} = v\cos(\psi)$$
$$\dot{Y} = v\sin(\psi)$$

$$\dot{\psi} = \omega = \frac{v\,\delta}{\ell_f + \ell_r}$$

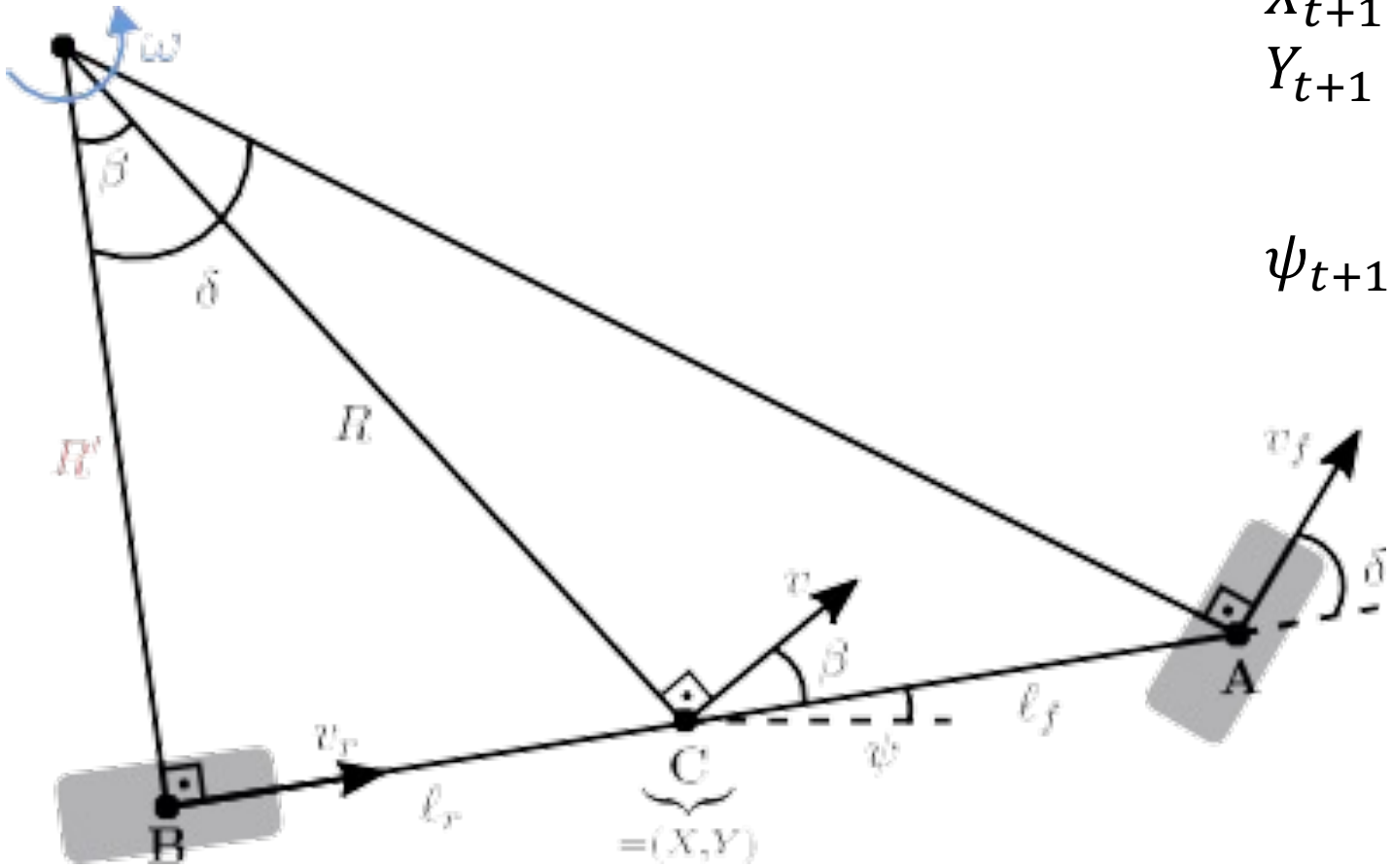
(assuming β and δ are small)



Kinematic Bicycle Time Discretized Model

Model

Rotation Center

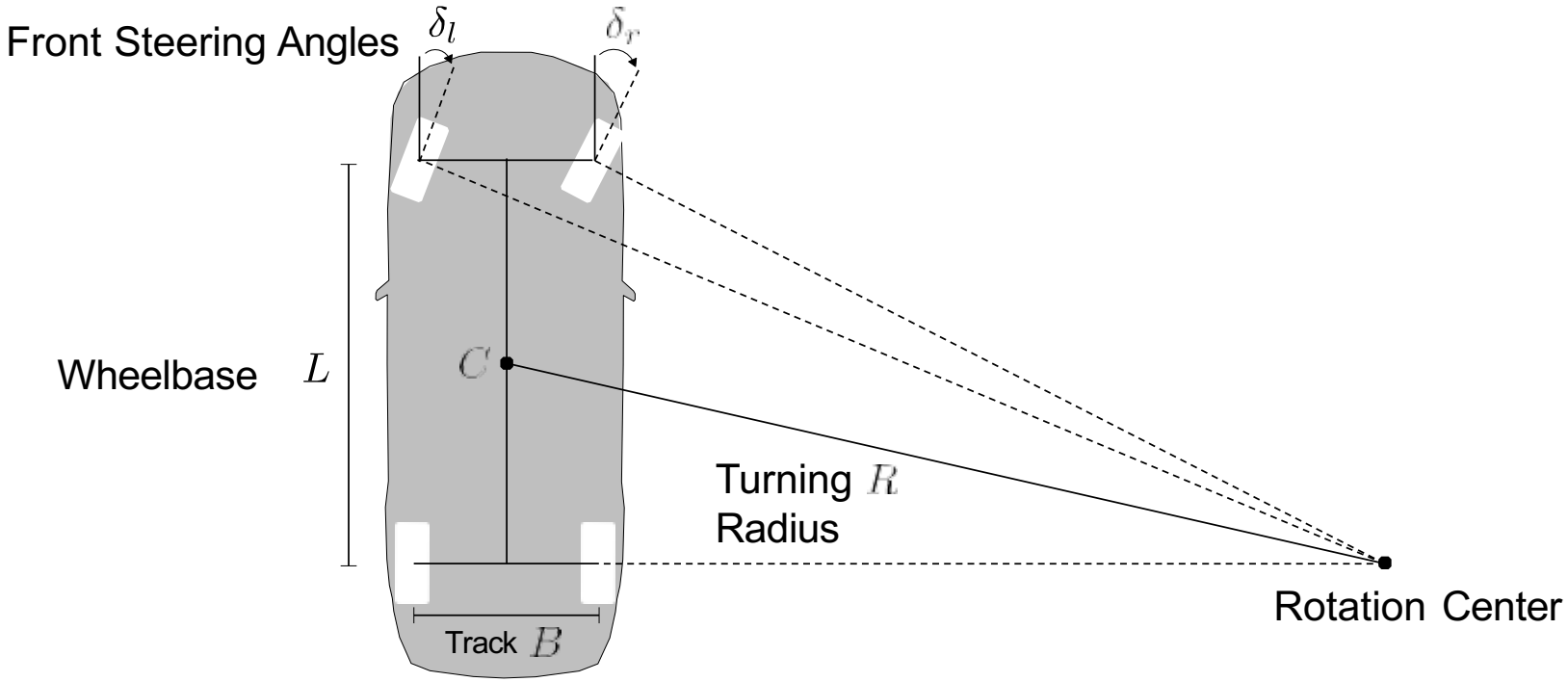


Motion Equations

$$X_{t+1} = X_t + v\cos(\psi_t)\Delta t$$
$$Y_{t+1} = Y_t + v\sin(\psi_t)\,\Delta t$$

$$\psi_{t+1} = \psi_t + \frac{v\,\delta}{\ell_f + \ell_r}\Delta t$$

Ackermann Steering Geometry

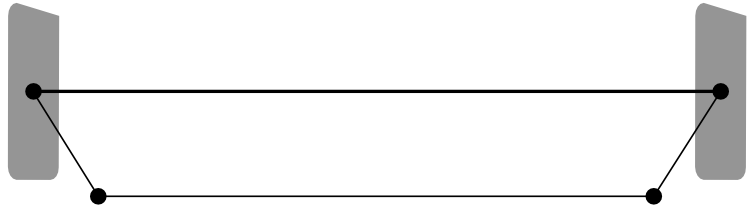


- In practice, the left and right wheel steering angles are not equal if no wheel slip
- Combination of admissible steering angles called Ackerman steering geometry
- If angles are small, the left/right steering wheel angles can be approximated:

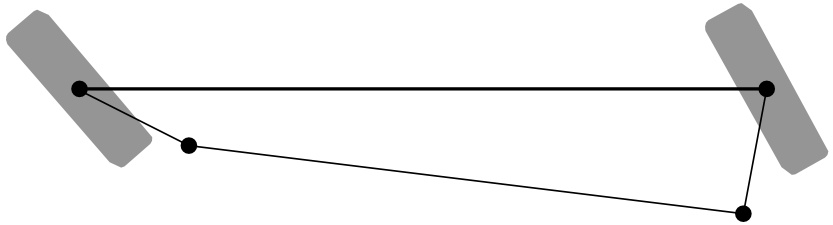
$$\delta_l \approx \tan\left(\frac{L}{R+0.5B}\right) \approx \frac{L}{R+0.5B} \qquad \delta_r \approx \tan\left(\frac{L}{R-0.5B}\right) \approx \frac{L}{R-0.5B}$$

Ackermann Steering Geometry

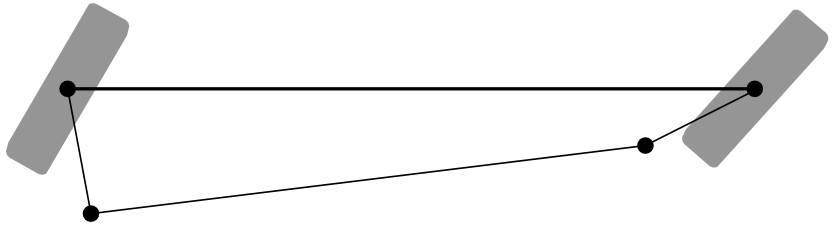
Trapezoidal Geometry



Left Turn



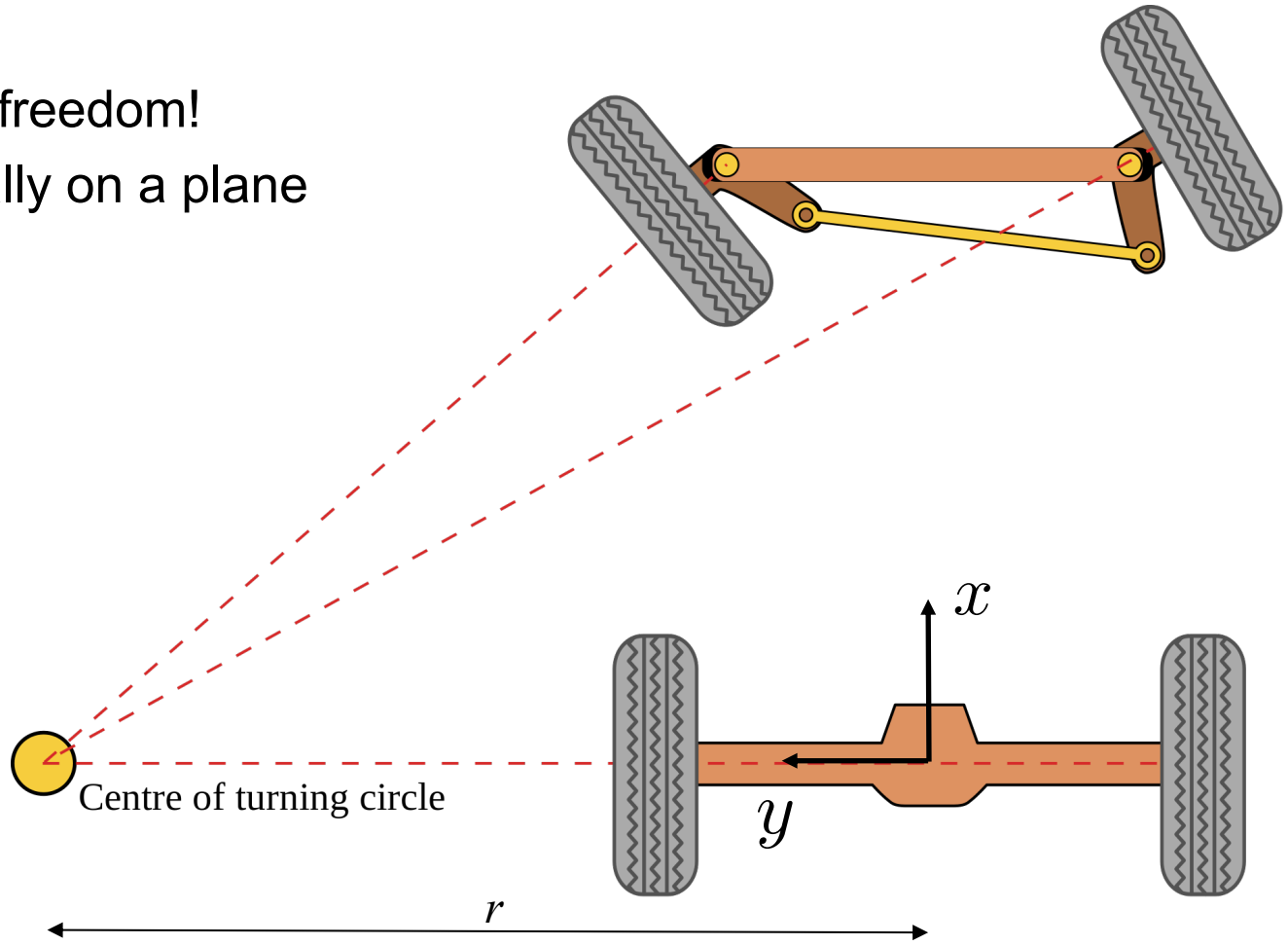
Right Turn



- In practice, this setup can be realized using a trapezoidal tie rod arrangement

Vehicle kinematic motion model

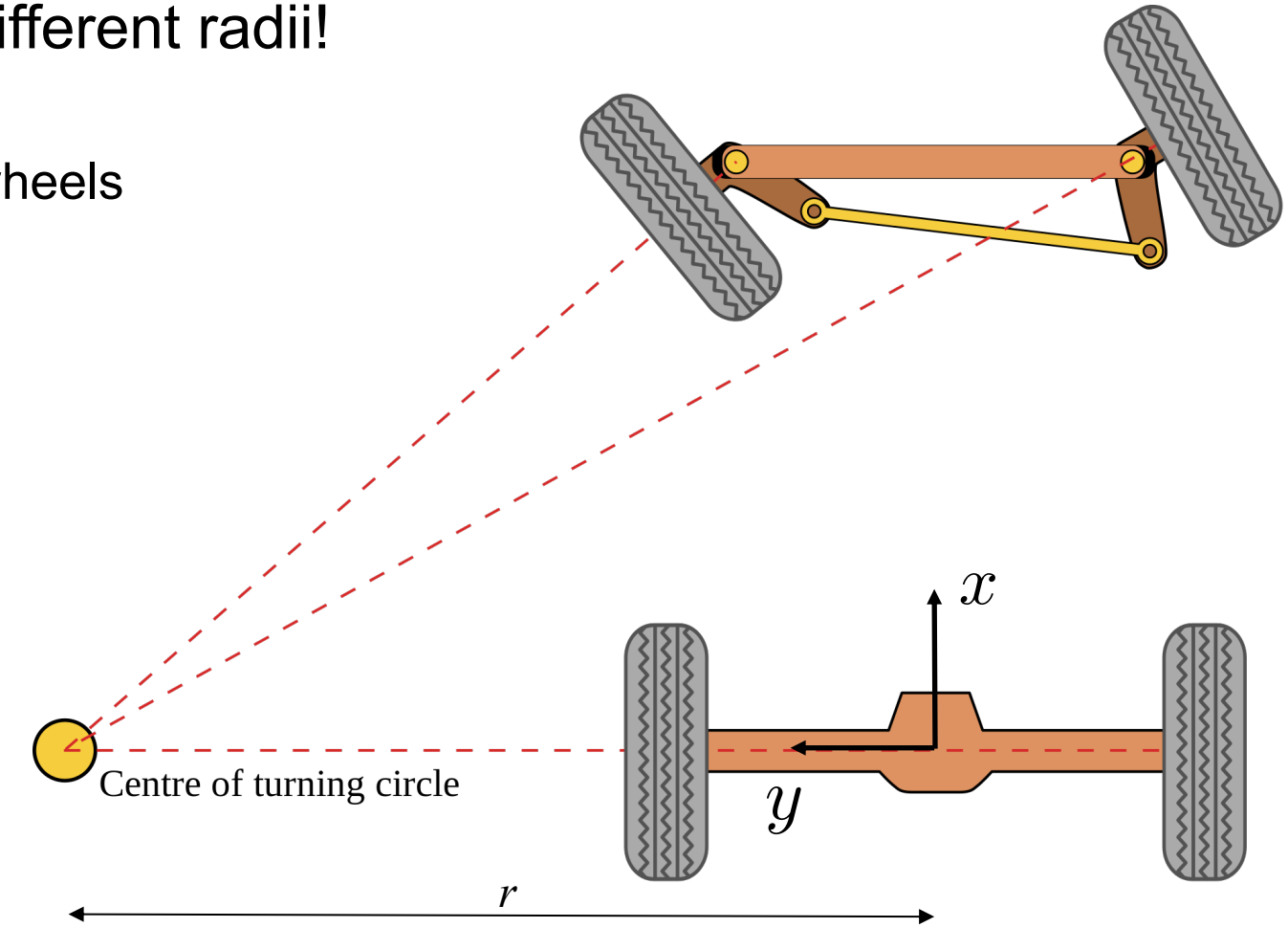
- Important insights:
  - The vehicle has only 2 degrees of freedom!
  - The vehicle's motion happens locally on a plane
    - 2 DoF translation displacement
    - 1 DoF rotation displacement
  - → Vehicle motion is exposed to non-holonomic constraints



Vehicle kinematic motion model

- Wheels drive on circles with different radii!
  - => wheels have different speeds!
  - Often: vehicle is driven on back wheels
  - => need a way to drive wheels with different speeds!
  - => Differential (gear box)

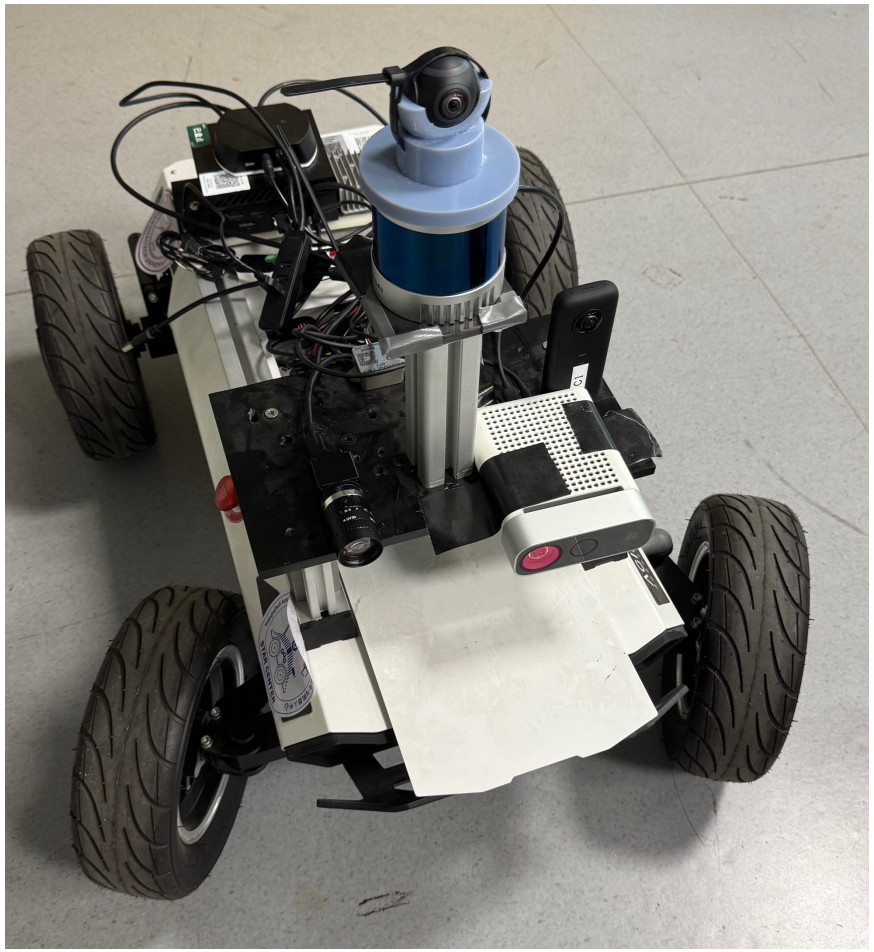
    != Differential drive robot!



https://www.youtube.com/watch?v=nC6fsNXdcMQ

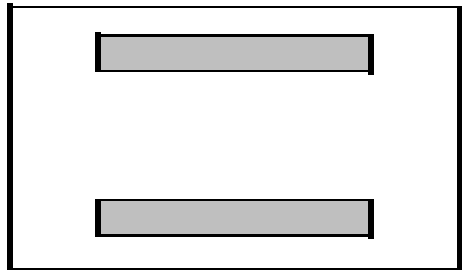
Example: AglieX Hunter SE

- Ackerman steering – with differential gearbox for the back wheels
- Good: No slipping when driving
- Really fast robot (4.8 m/s)
- Bad: Slipping when driving that fast and braking/ taking turns!
- Need to take its mass into account when controlling it => not just Kinematics but Dynamics for control!



Example: Clearpath Jackal Mobile Base

- Differential drive robot: Can control left and right wheels independently
- Bad: Slipping when turning



- Good: Forward and inverse kinematics easy: can turn on the spot!
- Maximum speed: 2 m/s

DYNAMIC BICYCLE MODEL

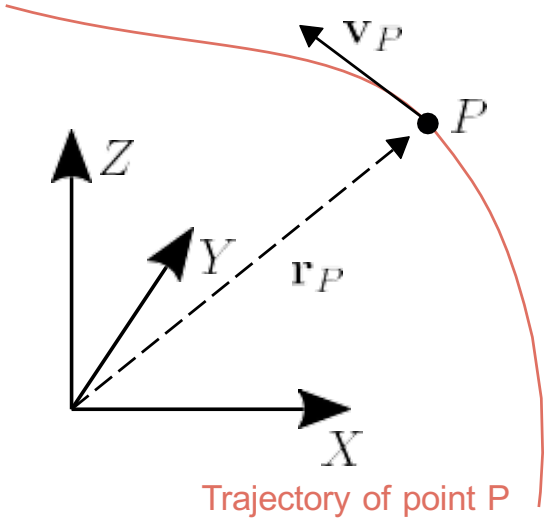
Dynamics of a Rigid Body

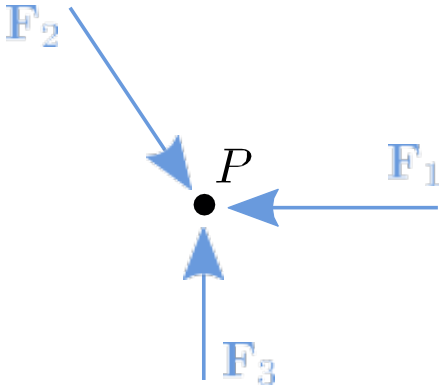
Translatory Motion of a Point:

- Consider **point** P with mass m in $\mathbb{R}^3$
- Let $\boldsymbol{r}_p(t) \in \mathbb{R}^3$ be its **position** in an inertial reference frame
- Let $\boldsymbol{v}_p(t)$ denote its **velocity** and $\mathbf{a}_p(t)$ its **acceleration**
- The **linear momentum** of P is defined as $\mathbf{p}_p(t) = m\mathbf{v}_P(t)$
- By **Newton's second law** we have

$$\frac{d}{dt}\mathbf{p}_p(t) = m\mathbf{a}_p(t) = F_{net}(t) = \sum_i \mathbf{F}_i(t)$$

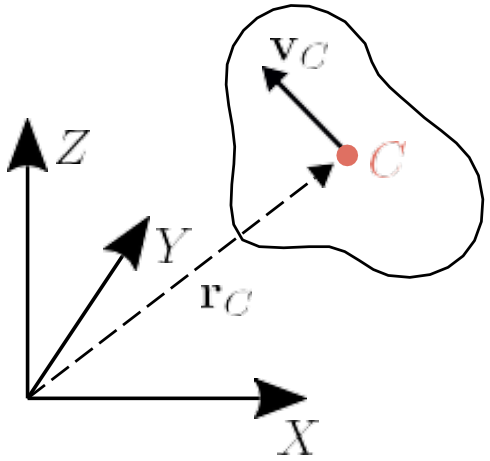
where $\mathbf{F}_i(t)$ represent all forces acting on the point mass P





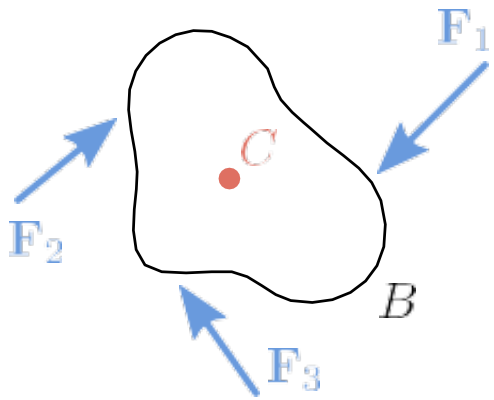
Dynamics of a Rigid Body

Translatory Motion of a Rigid Body:



- Consider a **rigid body** B with mass m in $\mathbb{R}^3$
- Let $\mathbf{r}_C(t) \in \mathbb{R}^3$ be the **position** of its **center of gravity C**
- Let $\mathbf{v}_C(t)$ denote its **velocity** and $\mathbf{a}_C(t)$ its **acceleration**
- The **linear momentum** of B is defined as $\mathbf{p}_B(t) = m\mathbf{v}_C(t)$
- The **center of gravity** of a rigid body **behaves like a point mass** with mass m and as if all forces act on that point

$$\frac{d}{dt}\mathbf{p}_B(t) = m\mathbf{a}_C(t) = F_{net}(t) = \sum_i \mathbf{F}_i(t)$$



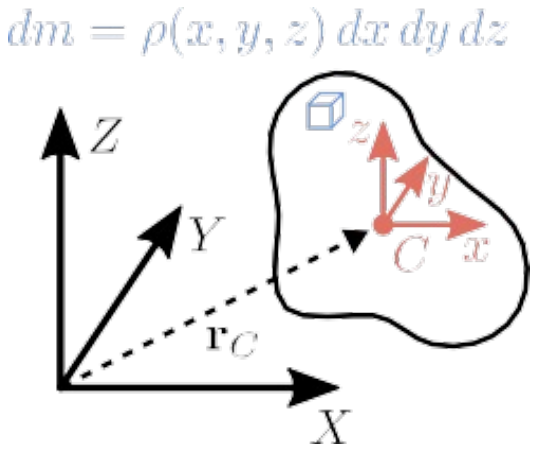
where $\mathbf{F}_i(t)$ represent all forces acting on the rigid body B

Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body:

- For the **rotatory motion,** also the geometric shape of B and the spatial distribution of its mass is important
- Let $\rho(x, y, z)$ be the **body's density function:**

$$m = \int_B \rho(x, y, z)\, dx\, dy\, dz = \int_B dm$$



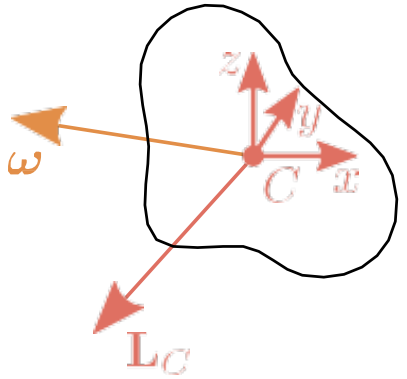
- The **inertia tensor** of B is defined as

$$\Theta = \begin{bmatrix} I_x & I_{xy} & I_{xz} \\ I_{yx} & I_y & I_{yz} \\ I_{zx} & I_{zy} & I_z \end{bmatrix}$$

$$\underbrace{\begin{aligned} I_x &= \textstyle\int_B (y^2 + z^2)\, dm \\ I_y &= \textstyle\int_B (x^2 + z^2)\, dm \\ I_z &= \textstyle\int_B (x^2 + y^2)\, dm \end{aligned}}_{\text{moments of inertia}} \qquad \underbrace{\begin{aligned} I_{xy} &= I_{yx} = -\textstyle\int_B xy\, dm \\ I_{xz} &= I_{zx} = -\textstyle\int_B xz\, dm \\ I_{yz} &= I_{zy} = -\textstyle\int_B yz\, dm \end{aligned}}_{\text{moments of deviation}}$$

Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body:



- Let $\boldsymbol{\omega}$ be the vector of **angular velocities:**

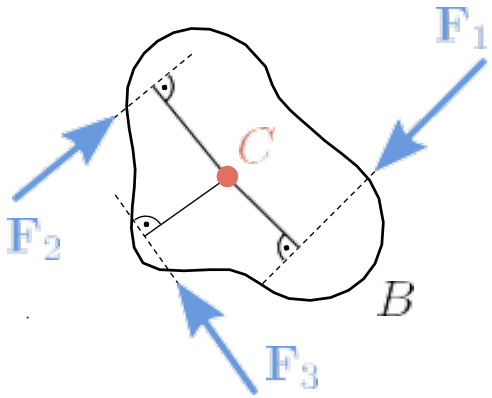
$$\boldsymbol{\omega} = (\omega_x \ \omega_y \ \omega_z)^T$$

- The **angular momentum** $\mathbf{L}_C$ of the rigid body B is given by

$$\mathbf{L}_C = \boldsymbol{\Theta}\ \boldsymbol{\omega}$$

- By the **angular momentum principle**

$$\frac{d}{dt}\mathbf{L}_C(t) = \boldsymbol{\Theta}\,\dot{\boldsymbol{\omega}} = \mathbf{M}_{net}(t) = \sum_i \mathbf{M}_i(t)$$



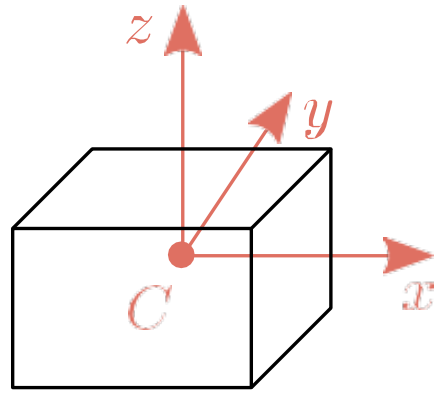
where $\mathbf{M}_i(t)$ are the moments of all forces acting on B with respect to the center of gravity C.

Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body with Canonical Coordinates:

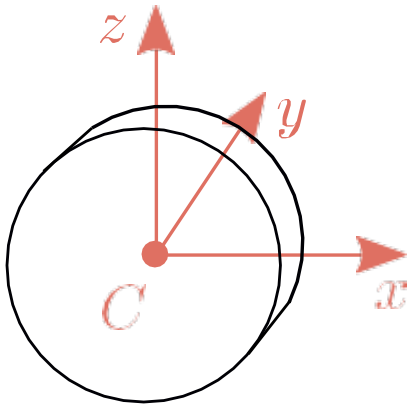
- If the body frame is chosen as a principal axis system for the rigid body (symmetry axes), the inertia tensor is diagonal:

$$\Theta = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$



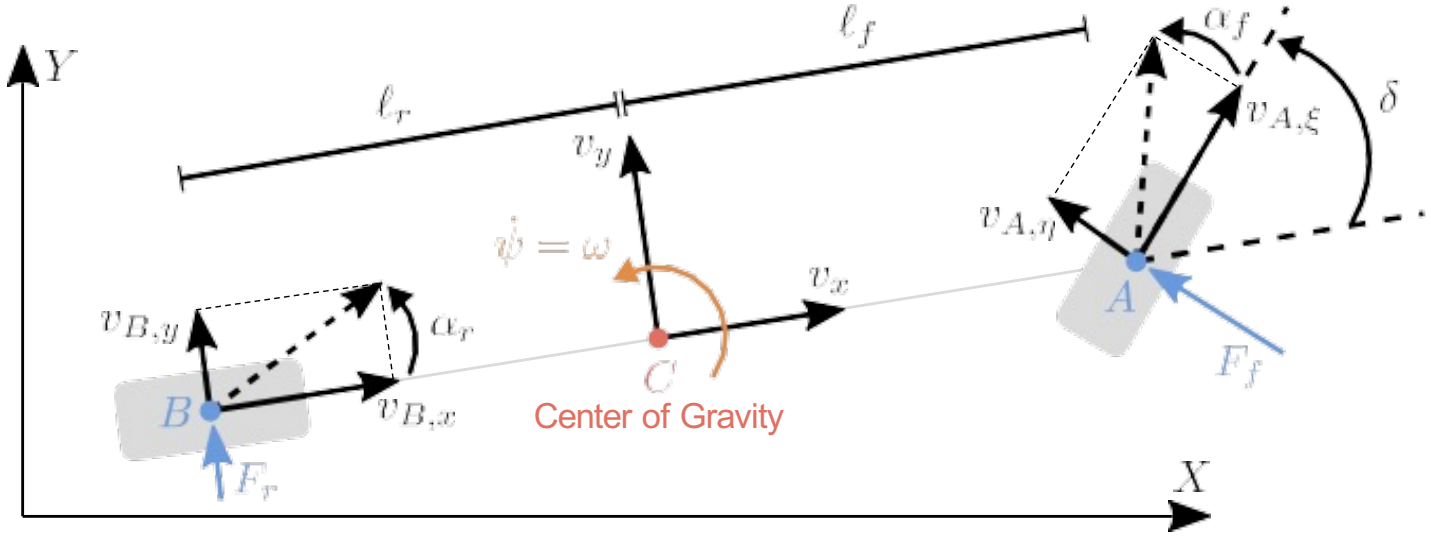
- For the planar motion of a rigid body in the x/y-plane:

$$\omega_x = \omega_y = 0 \quad \text{and} \quad M_x = M_y = 0$$



- Hence the angular momentum becomes $L_z = I_z\, \omega_z(t)$ and the angular momentum principle yields $I_z\, \dot{\omega}_z = \sum_i M_i$

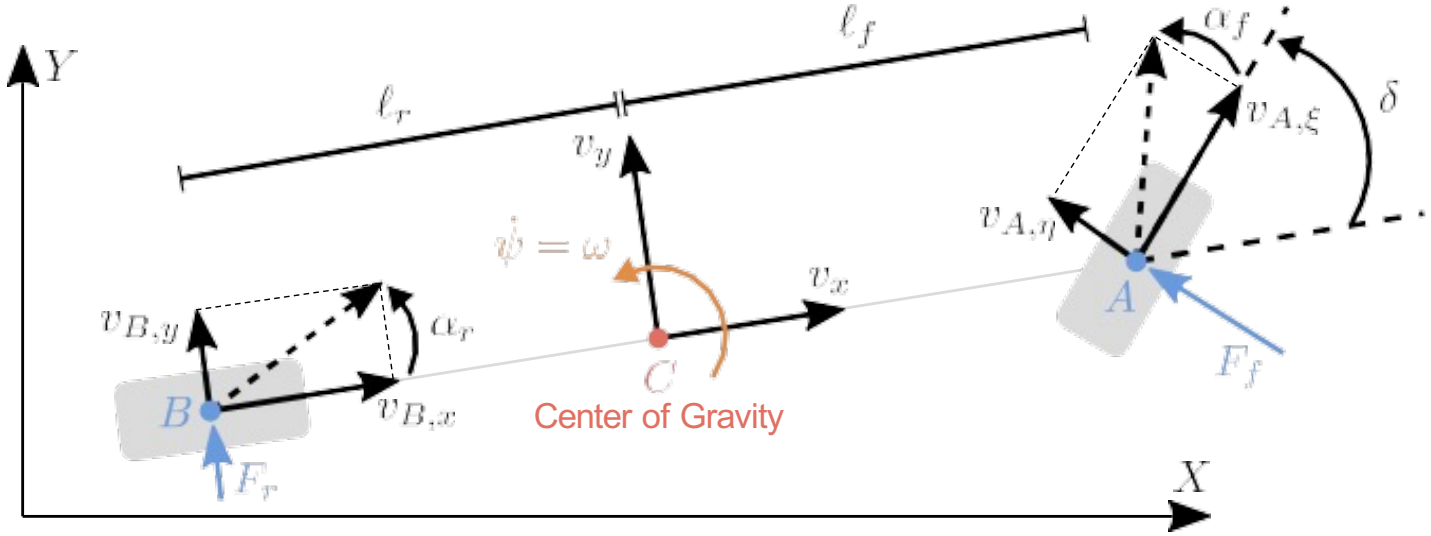
Dynamics of a Rigid Body



Assumptions:

- The vehicle's motion is restricted to the X/Y plane
- The vehicle is considered as a rigid body
- Only lateral tire forces, generated by a linear tire model
- Small steering angle δ: $\sin\delta \approx \delta \quad \tan\delta \approx \delta \quad \cos\delta \approx 1$
- Constant longitudinal velocity v_x

Dynamics of a Rigid Body



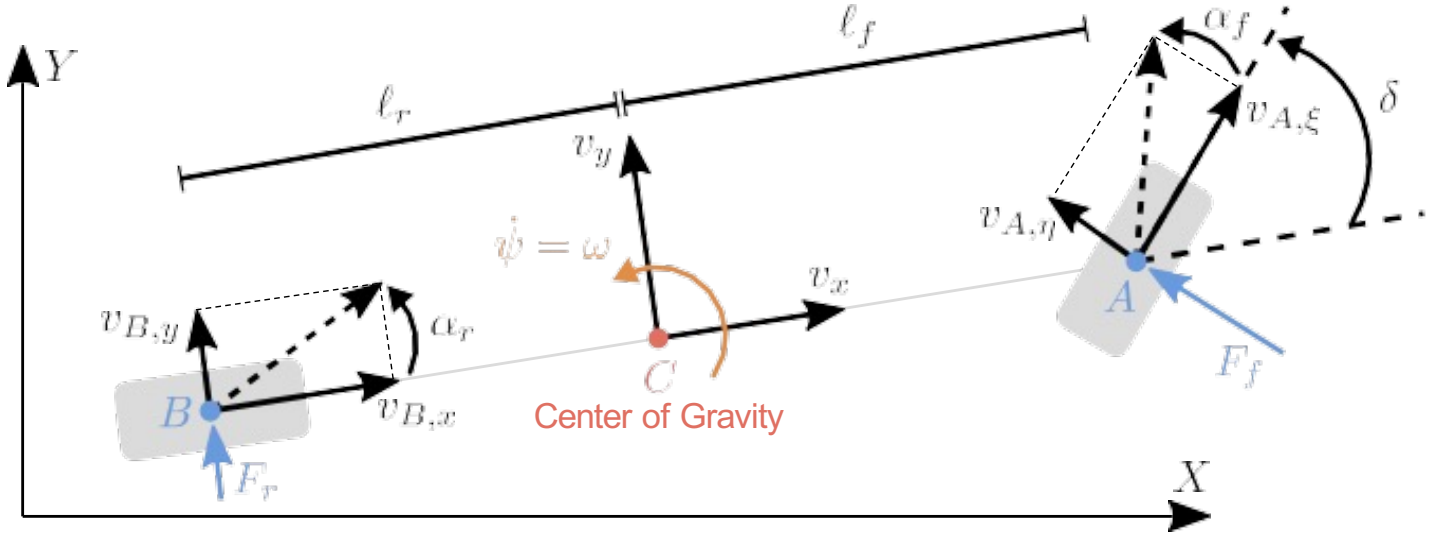
Lateral Dynamics:

$$ma_y = \sum_i F_{y,i} = F_r + F_f \cos\delta \approx F_r + F_f$$

$$a_y = \dot{v}_y + \omega\, v_x \qquad (\omega\, v_x = \text{centripetal acc.})$$

$$\Rightarrow \quad m(\dot{v}_y + \omega v_x) = F_r + F_f$$

Dynamics of a Rigid Body

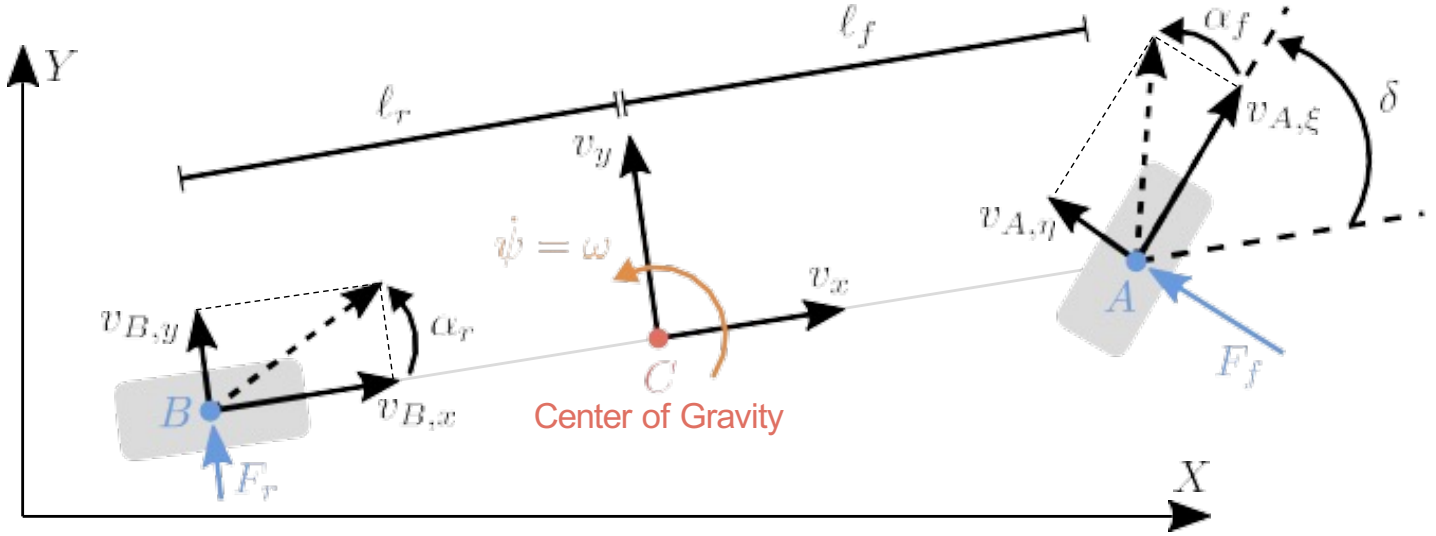


Yaw Dynamics:

$$I_z \dot{\omega} = \sum_i M_i = -l_r F_r + l_f F_f \underbrace{\cos\delta}_{\approx 1}$$

$$\Rightarrow \quad I_z \dot{\omega} = -l_r F_r + l_f F_f$$

Dynamics of a Rigid Body



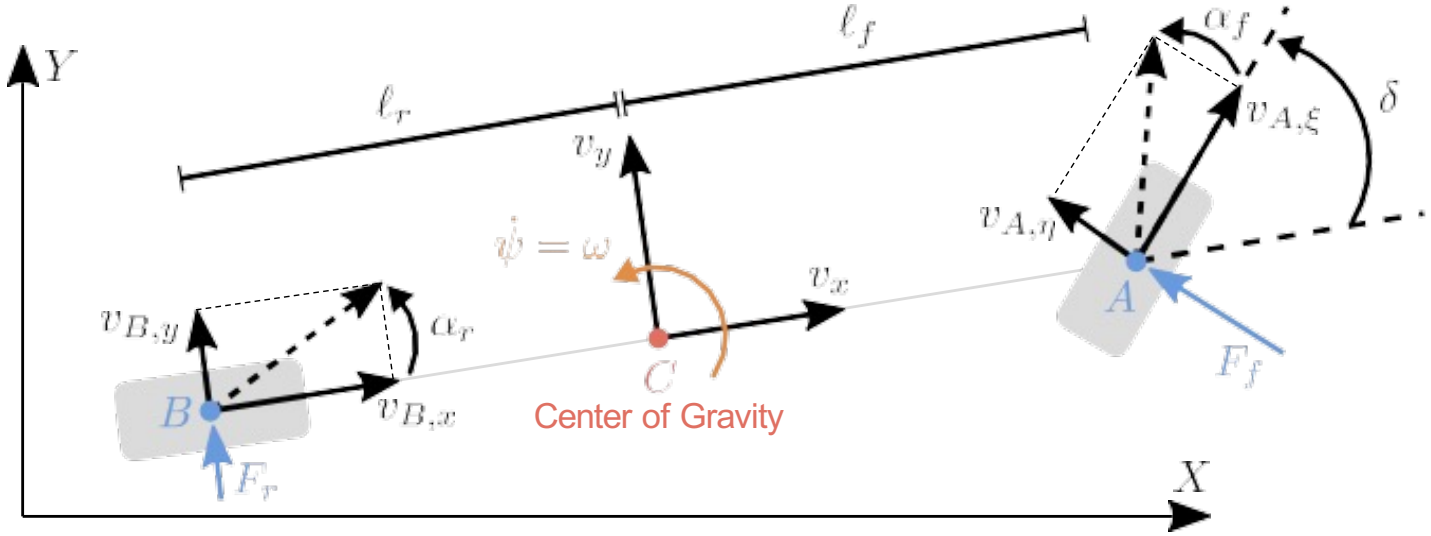
Tire Forces:

$$F_r = -c_r \alpha_r \approx -c_r \tan(\alpha_r) = -c_r \frac{v_{B,y}}{v_{B,x}} \qquad F_f = -c_f \alpha_f \approx -c_f \tan(\alpha_f) = -c_f \frac{v_{A,\eta}}{v_{A,\xi}}$$

$$v_{B,x} = v_x \quad v_{B,y} = v_y - \omega l_r \qquad v_{A,x} = v_x \quad v_{A,y} = v_y + \omega l_f$$

$$v_{A,\xi} = v_{A,x} \underbrace{\cos(\delta)}_{\approx 1} + v_{A,y} \underbrace{\sin(\delta)}_{\approx \delta} \qquad v_{A,\eta} = -v_{A,x} \underbrace{\sin(\delta)}_{\approx \delta} + v_{A,y} \underbrace{\cos(\delta)}_{\approx 1}$$

Dynamics of a Rigid Body



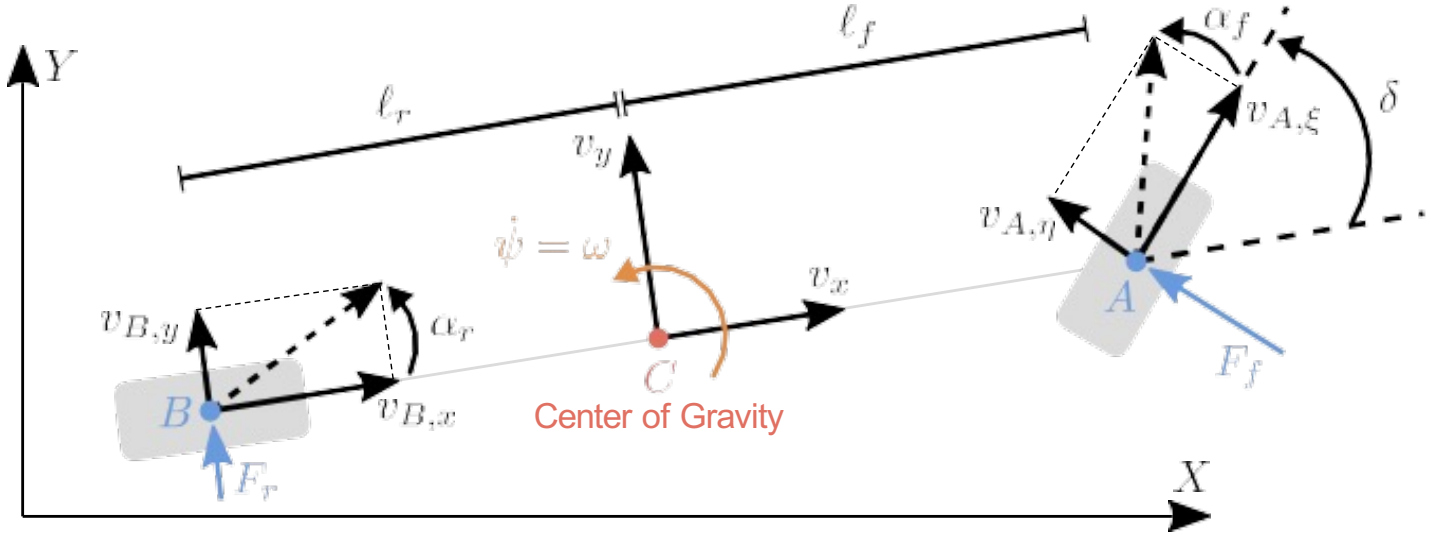
Tire Forces:

$$F_r = -c_r \frac{v_{B,y}}{v_{B,x}} = -c_r \frac{v_y - \omega l_r}{v_x}$$

$$F_f = -c_f \frac{v_{A,\eta}}{v_{A,\xi}} = -c_f \frac{-v_x\delta + v_y + \omega l_f}{v_x + (v_y + \omega l_f)\delta} \approx c_f\delta - c_f \frac{v_y + \omega l_f}{v_x}$$

Last approximation due to: $v_x \gg (v_y + \omega l_f)\delta$

Dynamics of a Rigid Body

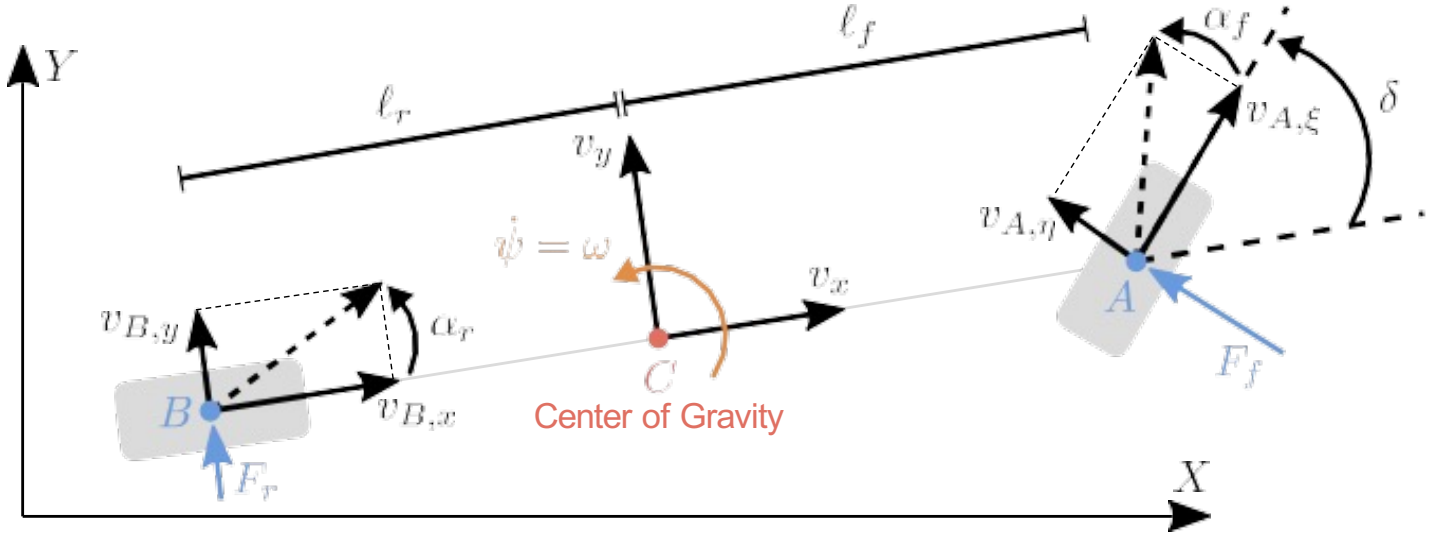


State Space Representation:

$$m(\dot{v}_y + \omega v_x) = \underbrace{-c_r \frac{v_y - \omega l_r}{v_x}}_{=F_r} + \underbrace{c_f \delta - c_f \frac{v_y + \omega l_f}{v_x}}_{=F_f}$$

$$I_z \dot{\omega} = -l_r \underbrace{\left(-c_r \frac{v_y - \omega l_r}{v_x}\right)}_{F_r} + l_f \underbrace{\left(c_f \delta - c_f \frac{v_y + \omega l_f}{v_x}\right)}_{=F_f}$$

Dynamics of a Rigid Body

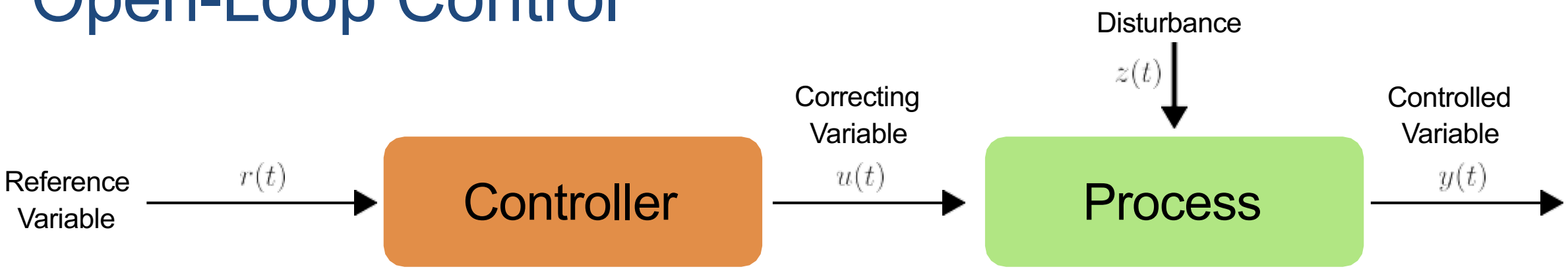


State Space Representation:

$$\begin{bmatrix} \dot{v}_y \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{c_r+c_f}{mv_x} & 0 & \frac{c_r l_r - c_f l_f}{mv_x} - v_x \\ 0 & 0 & 1 \\ \frac{l_r c_r - l_f c_f}{I_z v_x} & 0 & -\frac{l_f^2 c_f + l_r^2 c_r}{I_z v_x} \end{bmatrix} \underbrace{\begin{bmatrix} v_y \\ \psi \\ \omega \end{bmatrix}}_{\text{State}} + \begin{bmatrix} \frac{c_f}{m} \\ 0 \\ \frac{c_f}{I_z} l_f \end{bmatrix} \underbrace{\delta}_{\text{Input}}$$

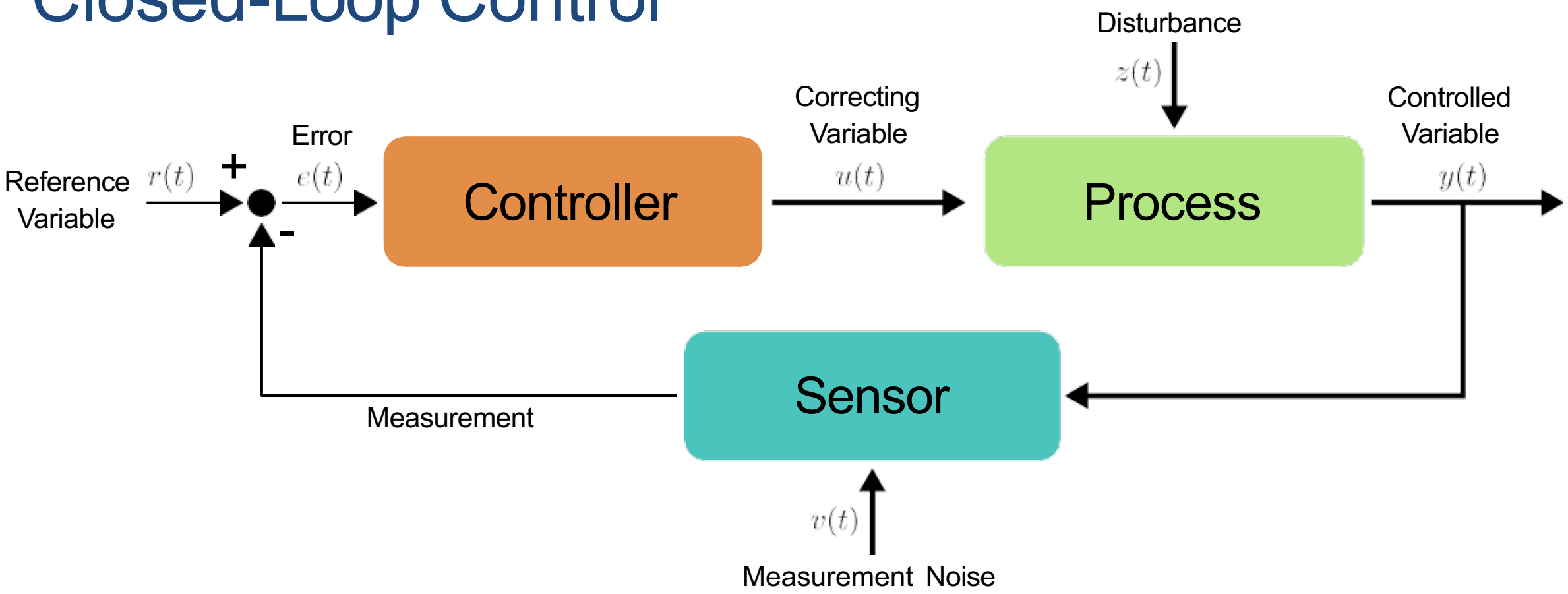
CONTROL

Open-Loop Control



- Requires **precise knowledge** of the plant and the influence factors
- **No feedback** about the controlled variable
- Cannot handle unknown disturbances, resulting in **drift**

Closed-Loop Control



- Exploit feedback loop to minimize error between reference and measurement

Centrifugal Governor

https://www.youtube.com/watch?v=B01LgS8S5C8

Closed-Loop Control

- We will be considering **closed-loop control** in this lecture
- A vehicle needs to be controlled both **longitudinally** and **laterally**
- We consider 3 different types of controllers:
  - **Black box controllers** don't require knowledge about the process
  - **Geometric controllers** exploit geometric relationships between the vehicle and the path, resulting in compact control laws for path tracking
  - **Optimal controllers** use knowledge of the system and minimize an objective function over future time steps

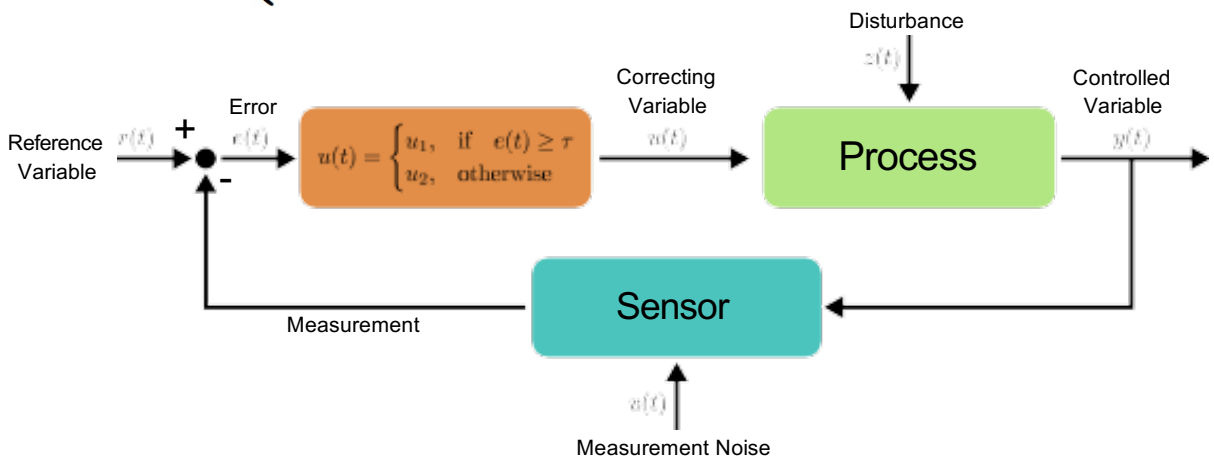
BLACK BOX CONTROL

Bang-Bang Control

Bang-Bang Control

- Also called: hysteresis controller
- Often applied, e.g. in household thermostats
- Switches abruptly between two states
- Mathematical formulation:

$$u(t) = \begin{cases} u_1, & \text{if} \quad e(t) \geq \tau \\ u_2, & \text{otherwise} \end{cases}$$



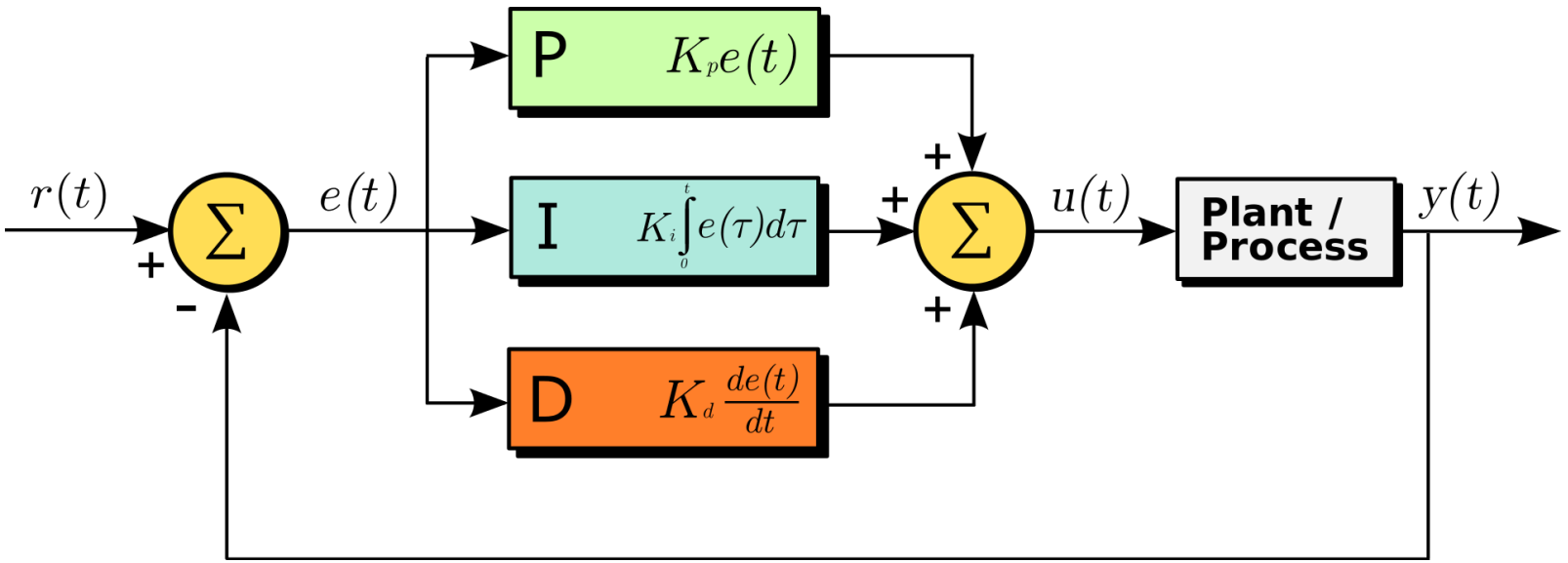
PID Control

PID: Proportional-Integral-Derivative Controller

- Input: Desired Speed (of wheel/ motor)
  - Actually: Error of the current speed (process variable) to the desired speed (setpoint)
- Output: Amount of power to the motor
- Not needed: Model of the plant process (e.g. motor, robot & terrain parameters)
- Parameters:
  - K_p proportional gain constant
  - K_i integral gain
  - K_d derivative gain
- Discrete Version:

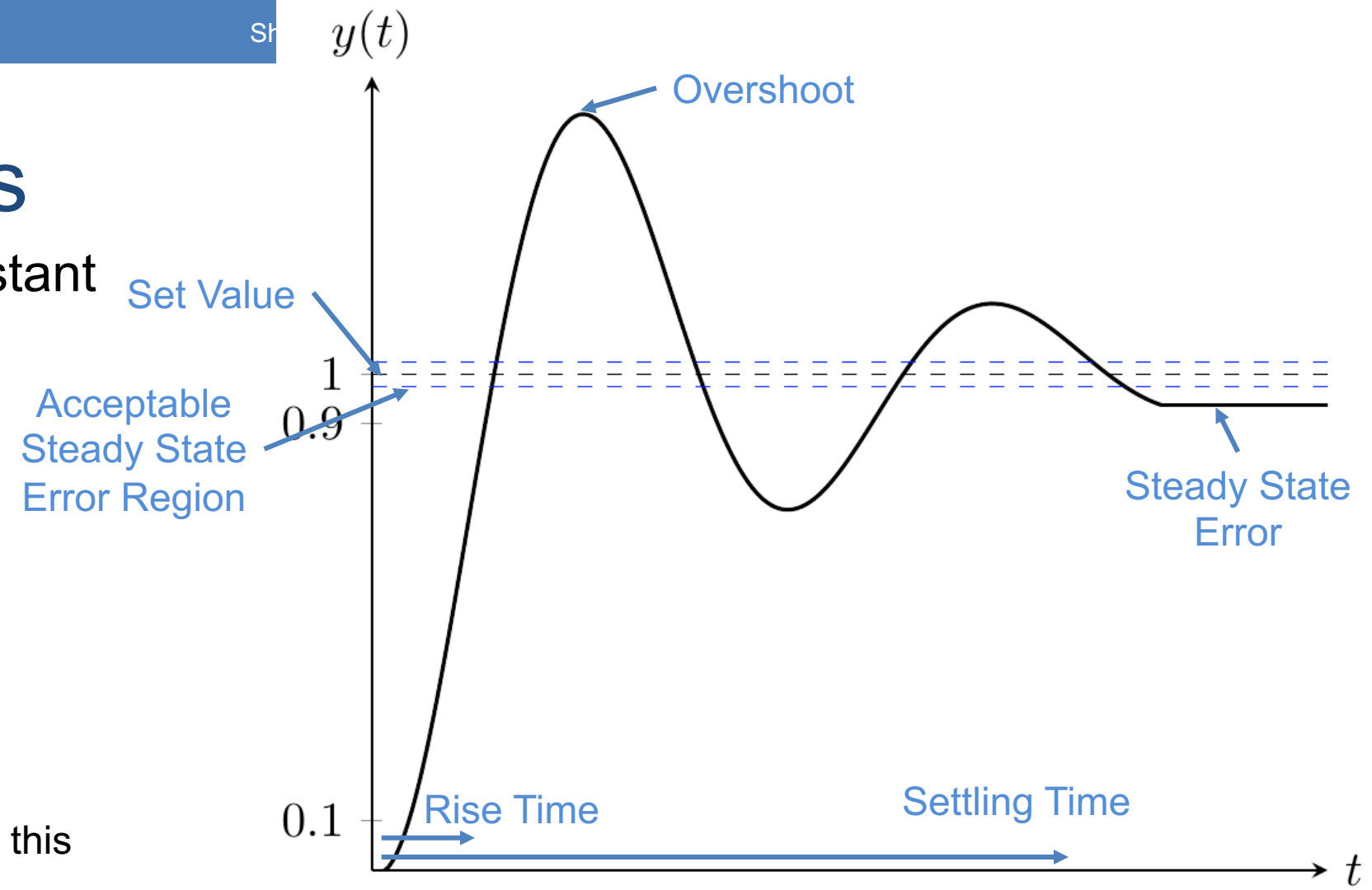
$$\int_0^{t_k} e(\tau)\,d\tau = \sum_{i=1}^{k} e(t_i)\,\Delta t$$

$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$



Tune Parameters

- K_p proportional gain constant
  - Too small: long rise time
  - Too big: big overshoot or even unstable control
  - Should contribute most of the output change
- K_i integral gain
  - Reduces steady state error
  - May cause overshoot
    - Leaky integration may solve this
- K_d derivative gain
  - Predicts error by taking slope into account
  - May reduce settling time and overshoot



Parameter Increase	Rise time	Overshoot	Settling Time	Steady-state error
Kp	↓	↑	Small Change	↓
Ki	↓	↑	↑	Great reduce
Kd	Small Change	↓	↓	Small Change

Table (2) PID controller parameter characteristics on a fan's response

https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/fas57_nyp7/Site/pidcontroller.html

Control Theory

- Other controllers used
  - P Controller
  - PD Controller
  - PI Controller

```
previous_error := 0
integral := 0

loop:
    error := setpoint - measured_value
    integral := integral + error × dt
    derivative := (error - previous_error) / dt
    output := Kp × error + Ki × integral + Kd × derivative
    previous_error := error
    wait(dt)
    goto loop
```

Pseudo Code PID Controller

- PID sufficient for most control problems
- PID works well if:
  - Dynamics of system is small
  - System is linear (or close to)
- Lots of Control Theory courses at ShanghaiTech University…

- Popular alternative: Model Predictive Control (MPC)
  - Optimal Control Technique: satisfy a set of constraints
  - Finite time horizon to look into the future (“plan”)
  - Used when PID is not sufficient; e.g.:
    - Very dynamic system
    - Second order system (oscillating system)
    - Multi-variable control
  - Use Cases: Chemical plants; planes; robot arms; legged robots; …

Control Hierarchy (Electrical Vehicle)

- Assume we have a goal speed
- Calculate needed wheel speeds using Kinematics =>
- Desired wheel speeds
  - Typically not just one wheel =>
  - Many motor controllers, motors, encoders
- Motor control loop
- Pose control loop

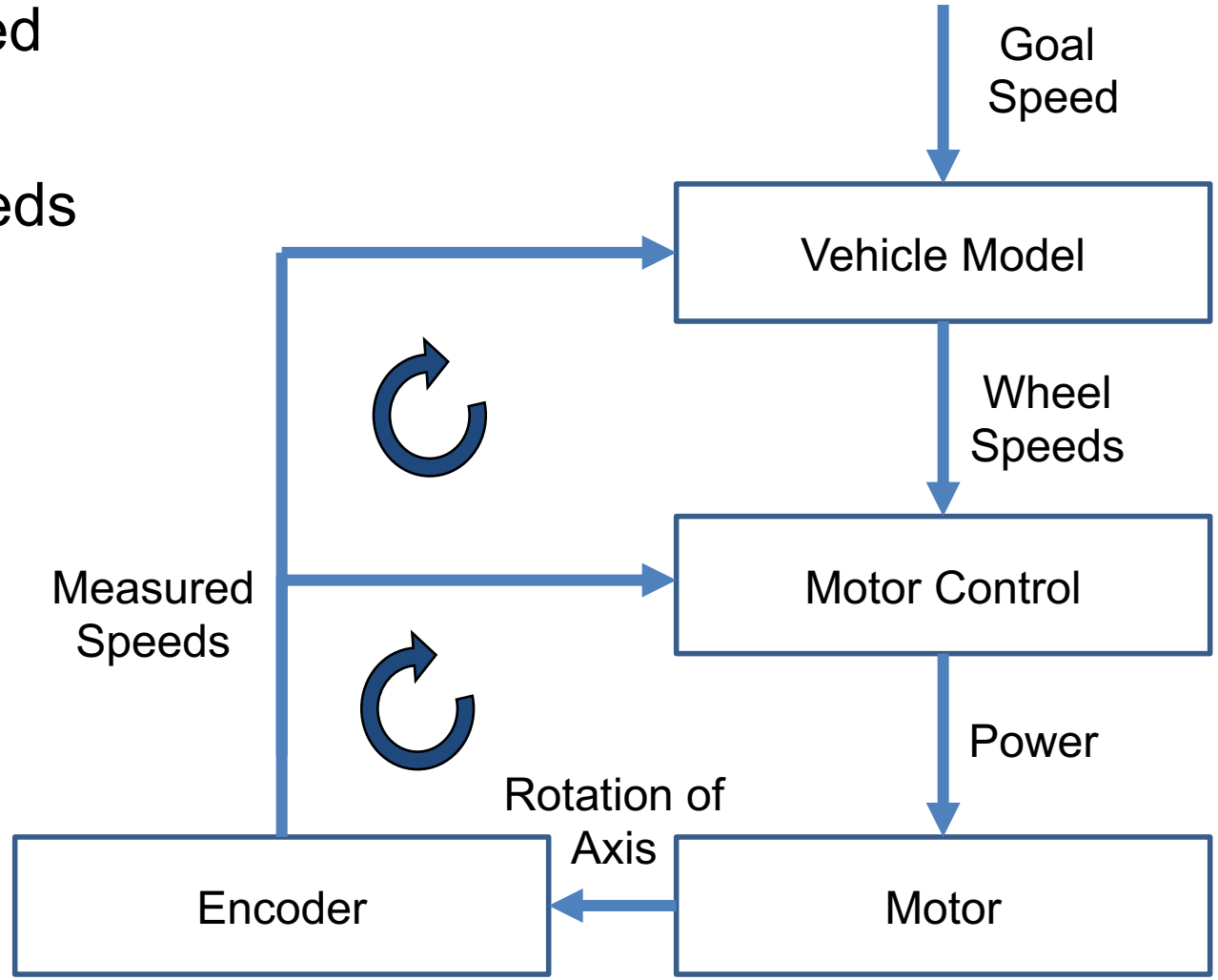
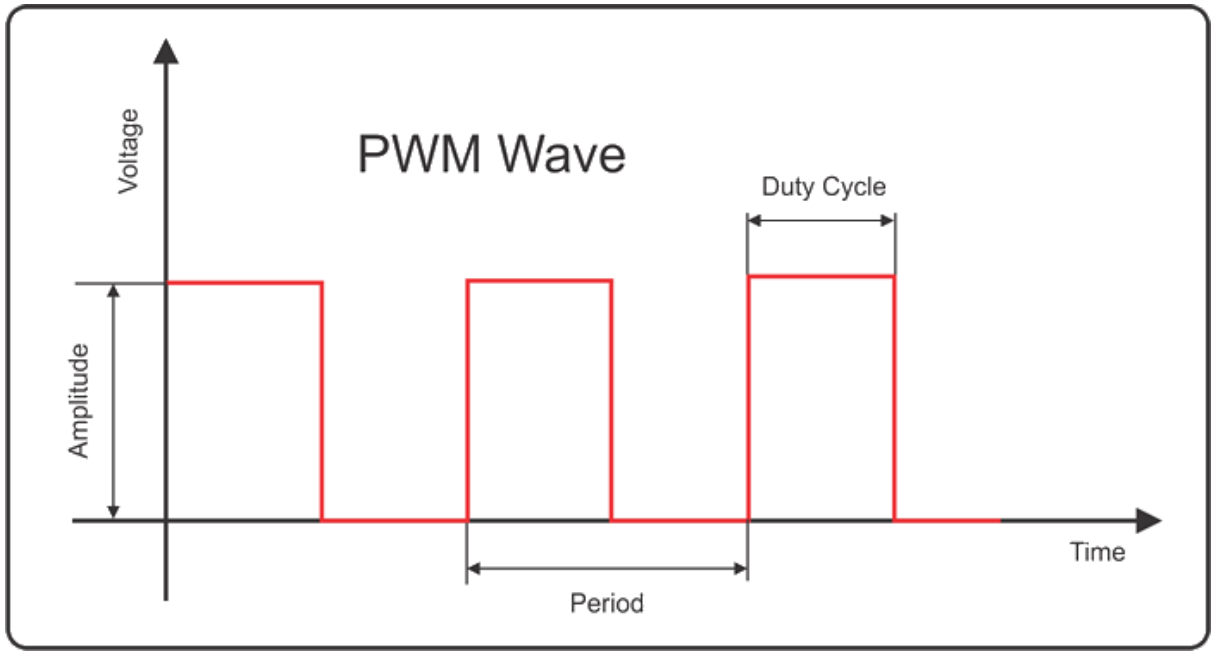
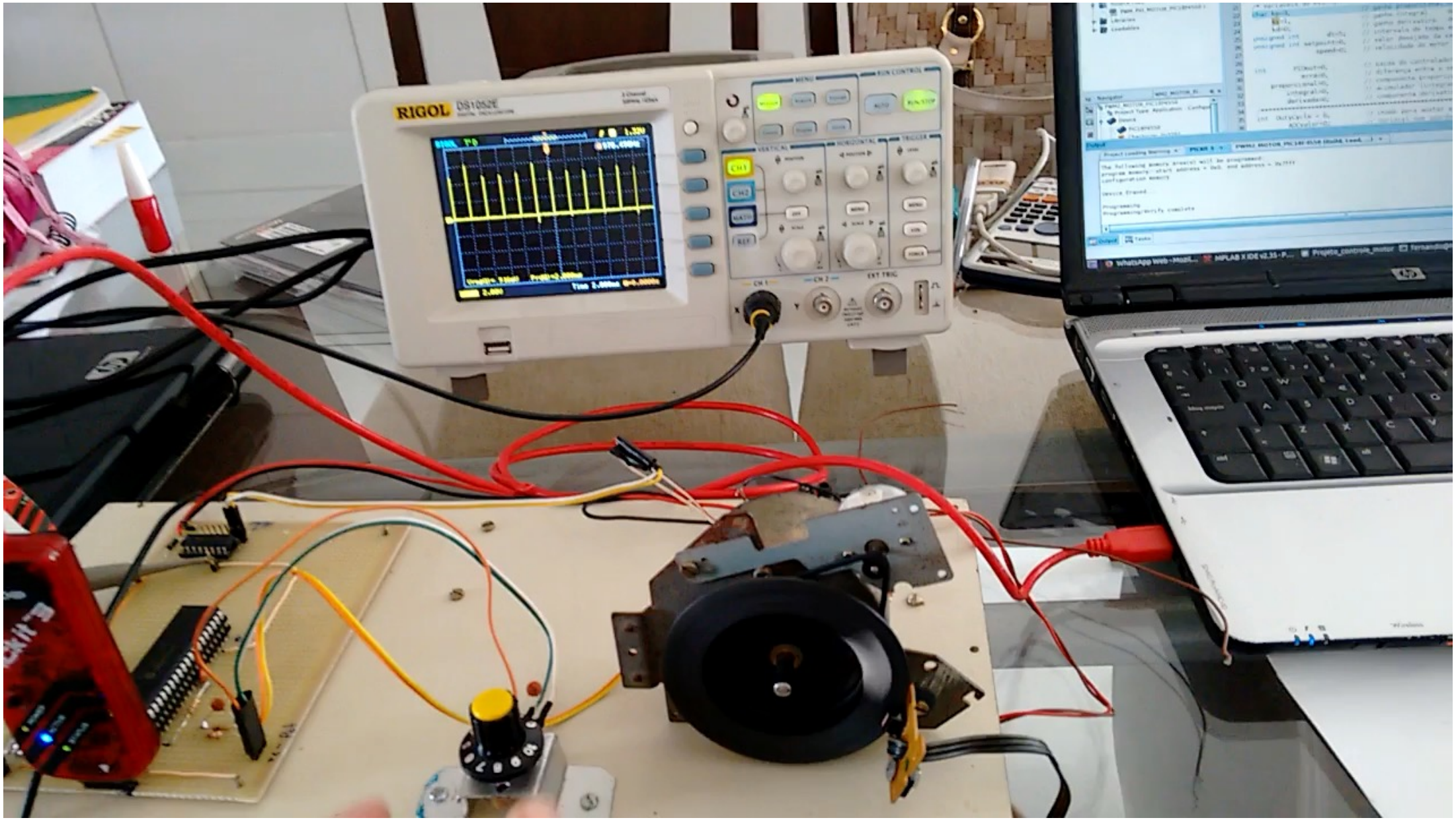


Image: zembedded.com

Pulse Width Modulation



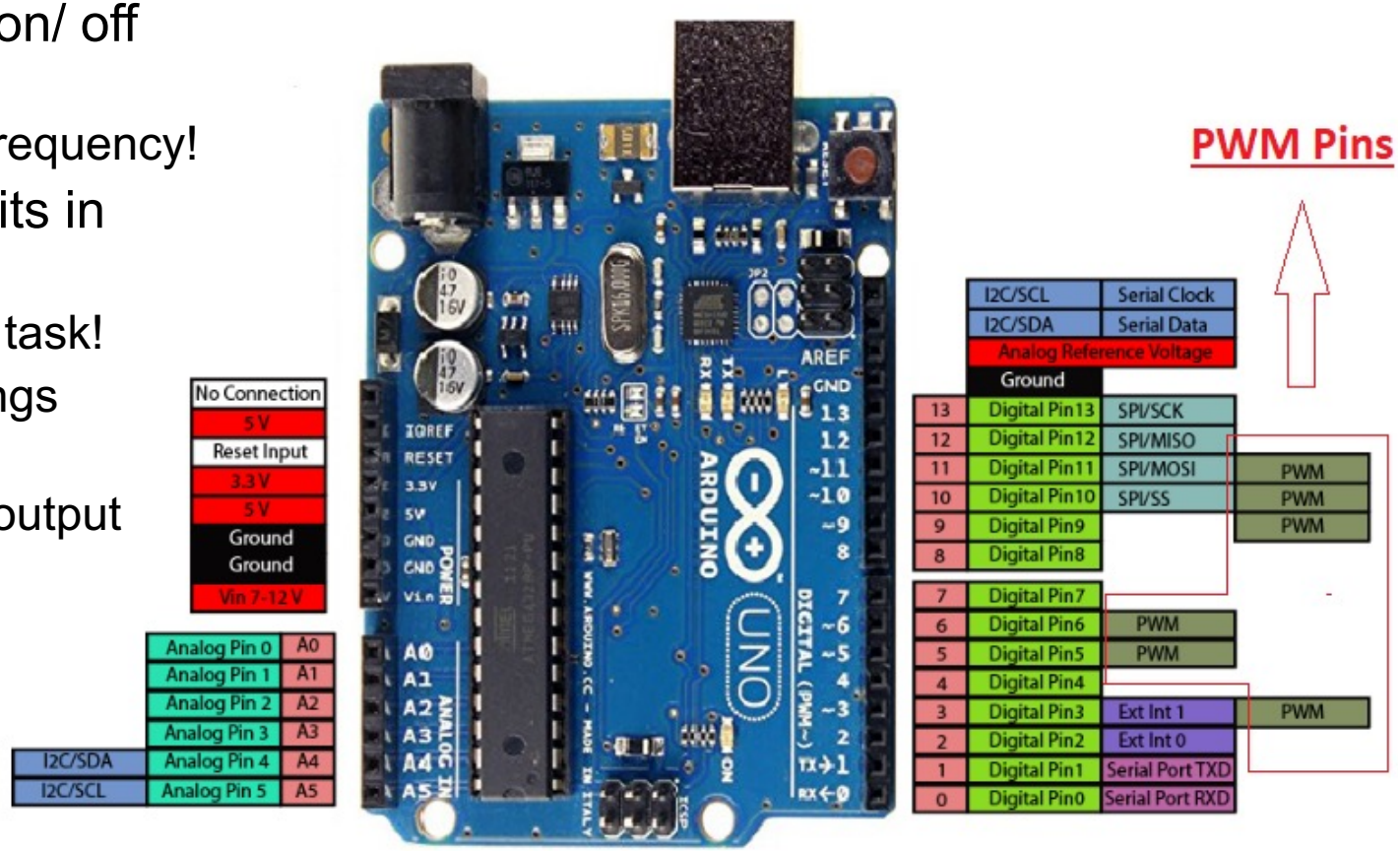
- How can Controller control power?
  - Cannot just tell the motor “use more power”
  - Output of (PID) controller is a signal
  - Typical: Analogue signal
- Pulse Width Modulation (PWM)
  - Signal is either ON or OFF
  - Ratio of time ON vs. time OFF in a given interval: amount of power
  - Frequency in kHz (= period less than 1ms)
  - Very low power loss
- Signal (typica 5V or 3.3V) to Motor Driver
- Used in all kinds of applications:
  - electric stove; audio amplifiers, computer power supply (hundreds of kHz!)



https://www.youtube.com/watch?v=4QzyG5g1blg

PWM Generation

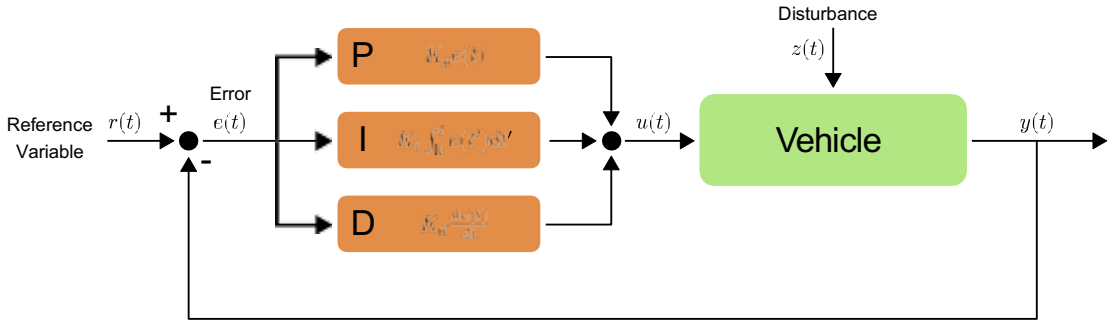
- Motor Control:
  - Frequency in kHz:
  - Smooth motion of motor wanted
  - Use inertia of the motor to smooth the on/ off cycle
    - Still: Sound of motor often from control frequency!
  - High frequency => use dedicated circuits in microcontroller to generate PWM!
    - CPU is not burdened with this mundane task!
    - CPU would suffer from inconsistent timings
      - Interrupts; preemptive computing
    - E.g. Arduino (ATmega48P) has 6 PWM output channels
    - Timer running independently of CPU
    - Comparing to a set register value – if it is up, the output signal is switched



Longitudinal Vehicle Control

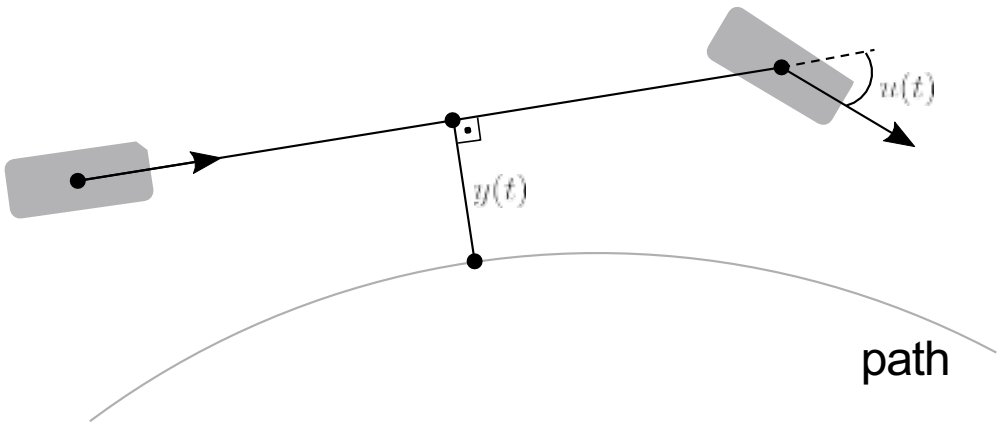
$$v(t) = v_{\max}\,(1 - \exp\,(-\theta_1\, d(t) - \theta_2))$$

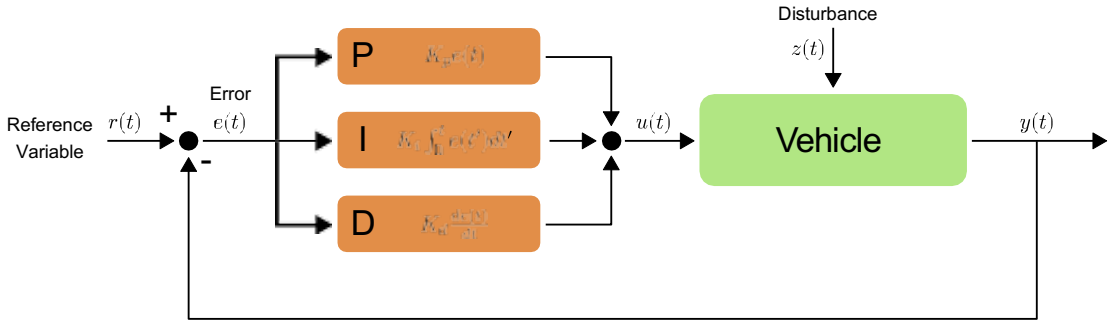
- $v(t)$: target velocity at time t
- $d(t)$: distance to preceding car



- Reference variable: $r(t) = v(t) =$ target velocity
- Correcting variable: $u(t) =$ gas/brake pedal
- Controlled variable: $y(t) =$ current velocity
- Error: $e(t) = v(t) - y(t)$

Lateral Vehicle Control





- Reference variable: $r(t) = 0 =$ no cross track error
- Correcting variable: $u(t) = \delta =$ steering angle
- Controlled variable: $y(t) =$ cross track error
- Error: $e(t) = -y(t) =$ cross track error

Controlling Self-Driving Cars

Aerospace Controls Laboratory

Massachusetts Institute of Technology

https://www.youtube.com/watch?v=4Y7zG48uHRo