Autonomous Vehicles Planning and Control

* Fill the gap between theory (equations) and coding
* Optimized to understand easily (Informal Writing Style, Omit the exception and unimportant and unusual things)
* Application oriented approach
* Theoretical background where necessary

# Part 1: Introduction

* Introduction Autonomous Vehicles
* Developing an Autonomous Vehicles
* Control Fundamentals

# Part 2: Application Examples

* Ground Vehicles
* Aerial Vehicles
* Marine Crafts

# Part 3: Modelling and Simulation

* System Modelling
* System Identification
* Environmental Modelling
* Linear and Nonlinear Model
* Control-oriented Model
* ROS – Gazebo – Unity – Simulink
  + UUVSimulator and Heron
  + RobotX Virtual Competition

# Part 4: Planning and Control

* Control (Robust, Nonlinear, Optimal, Intelligent)
  + PID
  + LQR
  + MPC
  + NMPC
  + Fuzzy Logic
  + Others
* Guidance
* Planning
  + Hybrid A\*
  + RRT\*
  + Dynamic Window
  + MPC
  + TO
* Prediction

# Part 5: Sensing

* Signal Processing (Filters, FFT)
* Localization
* Mapping and SLAM
* Perception (CV)

# Part 6: Basic Control Theory

* Classical Control Theory
* Stability Analysis
* Mathematics for Control
  + Linear Algebra
  + Fourier Transform
  + Laplace Transform

# Part 7: Programming

* Programming Fundamentals
* Software Libraries

# Part 8: Machine Learning in Control

* Q-Learning
* Deep Q-Learning
* Reinforcement Learning

# Control System Topics

* Model-based Design (System Model, Environment Model, etc.)
* Filters
* Delay
* LTI system
* Transfer Function
* State Space
* Stability (Gain, Phase, Disc, Robust Control, Lyapunov Function)
* Minimum Phase System

# Map of Control Theory

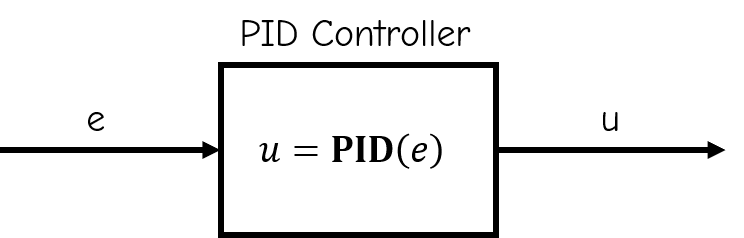
* Control Methods
  + Linear
    - PID
    - FSF
    - Lead-Lag
    - Loop-shaping
  + Nonlinear
    - Bang-Bang
    - Gain-scheduling
    - SMC
    - BS
  + Robust
    - H-infinity
    - Lyapunov-based
  + Optimal
    - LQR
  + Predictive
    - MPC
    - Robust MPC
    - NMPC
  + Adaptive
    - Model Reference Adaptive Control
    - Model Identification Adaptive Control
    - Extremum-seeking
    - Iterative Learning Control
    - Gain-scheduling
    - Multiple Models
    - Adaptive Pole Placement
    - Internal Model Control (IMC)
  + Intelligent
    - Fuzzy
    - NN
    - Neuro-fuzzy
    - Expert Systems
    - GA
    - Evolutionary Computation
    - Bayesian Control
    - ML
    - RL
* System Analysis
  + Block Diagram
  + Stability
  + Safety Margins
  + Robust Stability
  + Bode Plot, Step, Impulse, Ramp Response etc.
  + Root-locus
  + Nyquist Plots
  + Nichols Chart
  + Gang of Six
  + Phase Plane
  + Lyapunov Stability
  + Controllability
  + Observability
  + Non-minimum Phase System
* Modelling and Simulation
  + Transfer Function
  + State Space (Linear and Nonlinear)
  + System Identification
  + Linearization
  + Minimum Realizations
  + Hybrid System
* State Estimation
  + Filtering
  + Sensor Fusion
  + Calibration
  + Observer
  + KF
  + Particle
  + Sigma-point
  + Tracking
* Planning
  + Mapping
  + Trajectory Optimization
  + Constraints
  + Holonomic, Non-holonomic Redundant

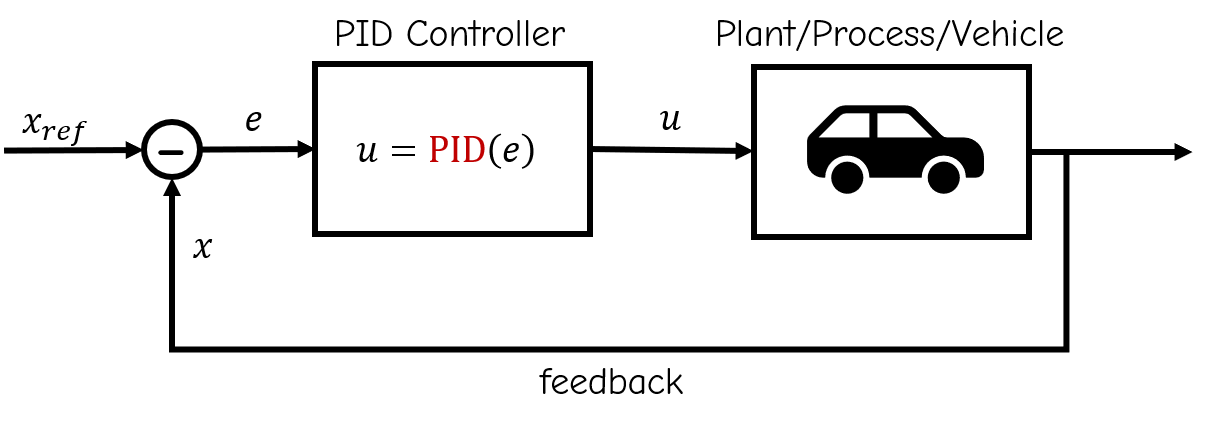
# Transfer Function in Code (4 Ways)

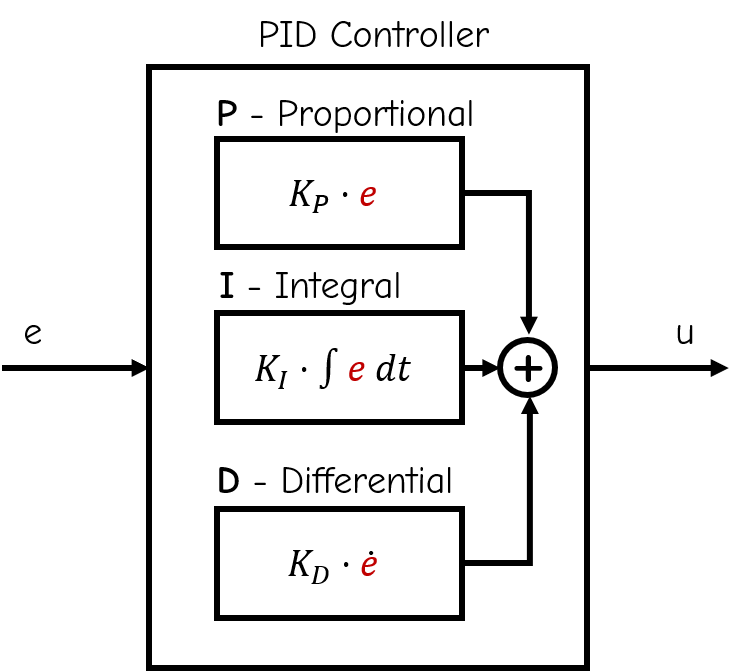
1. Numerically solve
   * TF to (higher) ODE 🡪 Inverse Laplace Transform
   * LHS of ODE 🡪 include derivatives of 🡪 use Integration to reduce to lower order derivative
   * RHS of ODE 🡪 may include derivatives of 🡪 use Derivative
2. TF to SS (Seems a good choice)
   * Need only Integration (No more Derivative for numerator “s”)
   * Slightly more accurate than 1st method, (no error associated with Derivative)
3. TF to z-domain
   * Method 1 and 2 sovle Continuous ODE, Discretization is done when coding (e.g Euler Integration, 1st order)
   * Z-domain 🡺 discretize the system frist
     1. When discretizing(converting to z-domain), Sampling time(time step) and Integration Method is already included.
     2. So, when writing code, no Integrator is needed.
     3. But, need to run the program in specified sampling time when using to z-domain
     4. If you want to change sampling time 🡪 do z-domain conversion again with new sampling time
   * Z-domain TF 🡪 rewrite in terms of
   * Take Inverse z-transform 🡪 when one is found, replace with one step unit deay of its multiplier. For 🡪 2 steps unit dealy, and so on.
   * The resulting equations are algebraic equations, that have to be updated using a specified time step. Compared to continuous one:
     1. less computationally expensive? (ODE vs Algebric equation)
     2. More accurate for relatively large (same) time step? (Continuous one’s accuracy depends on size of time step, while time step is already specified when discretizing for discrete version)
   * Side Node: the gain values of continuous PID are different from the discrete PID
4. MATLAB/Simulink (AutoCode)
   * C code generation
   * Has additional codes for robustness of the code
   * Easy to choose Integration method
   * Readibility of code is not quite good.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Control Method** | **Optimal** | **Non-linear** | **Robust** | **MIMO** | **Model-based** | **Remarks** |
| PID | 🌕 | 🗴 | 🌕 | 🗴 | 🗴 | Foundation of other controls  Optimality, Robustness depends on tuning  Other modifications could be done for non-linear, MIMO and model-based |
| FSF(PP) | 🗴 | 🗴 | 🗴 | 🗸 | 🗸 |  |
| LQR | 🗸 | 🗴 | 🗴 | 🗸 | 🗸 | Quadratic cost of weighted and |
| SFL | 🗴 | 🗸 | 🗴 | 🗸 | 🗸 | Not a controller itself, used with linear controllers |
| BS | 🗴 | 🗸 | 🌕 | 🗸 | 🗸 | In the steps, various controller could be used |
| SMC | 🗴 | 🗸 | 🗸 | 🗸 | 🗸 |  |
| H∞ | 🗸 | 🗴 | 🗸 | 🗸 | 🗸 | Robustness for bounded uncertainty  Cross-coupled MIMO |
| MPC | 🗸 | 🗴 | 🗸 | 🗸 | 🗸 | Quadratic Cost  Linear Constraints on and  Computationally expensive |
| NMPC | 🗸 | 🗸 | 🌕 | 🗸 | 🗸 | Nonlinear Cost  Nonlinear Constraints on and  Computationally very expensive |

# PID Control







* What is PID Control?
* How does it work?
* Mathematical Expression/Derivation/Equations
* Code for pure PID
* How to choose the PID Gains? (Tuning)
* Basic Modifications to PID
  + Saturation (Control Bounds)
  + Control Rate Bounds
  + Integral Anti-windup
  + Filtered Derivative and Direct use of signal rate
  + Set-point filtering
  + Use of and instead of and
  + Gain-scheduled PID
* Code for PID with some modifications
  + Function Approach
  + Class (Object) Approach
* Major Modifications
  + Feedforward PID
  + Cascaded PID
  + Set-point Weighting
  + Fractional PID
* PID Limitations
* PID Alternatives
* Other Notes for PID
* Feedback
* Continuous
* Most popular
* SISO (error in control out)
* Weighted sum of 3 terms

|  |  |  |  |
| --- | --- | --- | --- |
| **P** | Proportional to the error | The larger the error, the larger the control action. | Eliminate the basic error |
| **I** | Proportional to the accumulated error | The more error is accumulated, the larger the control action. | Eliminate the residual steady-state error (steady disturbance) (found in P only control). If the current control action is not enough to bring error to zero, the control action will be increased as time passes. |
| **D** | Proportional to rate of change of error | The more rapid the error changes, the greater the damping effect. | Reduce the overshoot and oscillation while keeping fast convergence. It will try to bring error rate to zero. Flatten the error trajectory into a horizontal line (so reduces overshoot) |

I term: Try to eliminate residual error left after applying P term

D term: estimate the future trand of error based on current its rate of change. The more rapid the change, the greater the damping effect.

|  |  |  |
| --- | --- | --- |
| **PID** | **:** | Proportional-Integral-Differential |
| **Function** | **:** | To eliminate the error  To stabilize a state variable at a desired set-point  To track a time-varying reference |
| **Remark** | **:** | Most popular control method |
| **Tuning Parameters** | **:** |  |

* Error 🡪

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  | |  |  | |
|  |  | To eliminate the (basic) error | |  | To eliminate **stead-state** error and **bias** | |  | To reduce **overshot** while keeping fast **convergence** | |
|  |  | Current Error | |  | Past Error | |  | Future Error | |

|  |  |
| --- | --- |
|  | **Code for PID Control** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **def** PID**(**error**,** error\_last**,** Integral**,** dt**,** PID\_Gains**,** u\_bound**):**  KP **=** PID\_Gains**[**0**]**  KD **=** PID\_Gains**[**2**]**  KI **=** PID\_Gains**[**1**]**  u\_min **=** u\_bound**[**0**]**  u\_max **=** u\_bound**[**1**]**  Proportional **=** KP**\***error  Integral **+=** KI**\***error**\***dt # Forward Euler  Integral **=** clamp**(**Integral**,**u\_min**,**u\_max**)** # Intergral Anti-Windup  Differential **=** KD**\*(**error**-**error\_last**)/**dt  u **=** Proportional **+** Integral **+** Differential  u **=** clamp**(**u**,**u\_min**,**u\_max**)**  **return** u**,** Integral |

## PID Extension

* Control Saturation/Bounds (Clamping between min and max)
* Intergral Anti-windup (to eliminate the accumulation of integral term over bounds when a steady-state error occurred for a long time) - 3 solution methods
  1. Disable Integration until control action has entered the allowable range
  2. Prevent the integral term from accumulating above and below bounds (of control action)
  3. Back calculating the integral term to constrain the output within feasible bounds. (If output is over the bound, substract the extra output from the integral term)
* Fitered Derivative (If the error signal has noise (e.g. measurement noise, sudden change of reference signal, the will be very large. To eliminate that, the should be filtered.
* Sometimes, instead of (rate of change of error), difference of rate of change of state(process) variable and that of reference or just (assume ) is also used. Because the derivatives of state variable is always continuous (no step change when ref is changed). Useful when is nicely measurable
* and : The idea is to make the controller more aggrasive or conservative by just changing one parameter . Chaning will affect all three terms.

## PID Modifications

* P only, PD, PI, I only (D only can not drive the error to zero, it derives the error rate to zero so the process signal become flat at set-point), PI: when derivative noise amplification is unacceptable

### Feedforward-PID

* Add feedforward term using the knowledge about the system(such as desired acceleration and inertia)
* FF value alone can often provide the major portion of controller output
* Since FF output is not affected by the process feedback, it can never cause the control system to oscillate, thus improving the system response with affecting stability (FF is open-loop)
* FF can be based on the setpoint and on extra measured disturbance
  + Total FF = Set-point FF + Disturbance FF
* Set-point weighting is a simple form of FF
* Set-point FF = invserse of Plant Dyanmics.
  + Plant Dyanmis:
  + FF Term: , to bontain (ref-y)=0 for perfect model G
  + But, Not all G can be inversed. Fit a causal and reliable model instead of inverting.
  + If the plant has delay reponse to control , has to be start earlier. But how much earlier? Delay has to be known.
* Disturbance FF
  + Plant will have two transfer functions: for process and for disturbance
  + Disturbance measurement for needed
* When FF term is included PID bandwidth can be lower (i.e. designed not be response relatively high frequency) 🡪 by doing so, some unwanted high frequency will be filtered (e.g. lower frequency noise), and the phase margin will be larger. But the PID controller will be slower (less responsive)
* With FF, PID need to only correct the errors caused by (FF)modelling errors and unmeasured disturbance. Some loads on PID are reduced, so PID can be tuned for more specific task instead of all kinds of error correction.

### Cascaded PID

* The control system is broken down into smaller problems.
* Outer-loop PID calculates the reference set-point for inner PID
  + Serve as a Refereance Generator (Low-pass filter)
* The inner PID controls the real acturators.
* Why? Time delays in the whole system is broken down into Slower component and Faster one.
* Slower one is controlled by outer-PID which has a long time constant. (slower response)
* Faster one in controlled by inner-PID which has a short time constant (faster reponse)
* E.g Bath Water temperature control
  + Outer-PID (Heater Temperature Generator)
    - Calculates the set-point temperature for heater
    - Error = Set-point bath temperature – measured bath temperature
  + Inner-PID (Heater Temperature Controller)
    - Calculate the heater’s power (gas value opening/electric power/On-Off state)
    - Error = set-point heater temperature – real(measured) heater temperature
  + The slow and fast dynamics are controlled by separate PIDs tuned for their best response
  + First Outer-PID serves as a Reference Generator/Low-pass filter to the inner-PID.

### Gain-sheduled PID

* Linearized the plant at a bunch of operating points
* Tune for each operation points, and store the gains in the database.
* Use them as:
  + If-then switching statement (not OK for many points)
    - May need low-pass filter to avoide sudden change
  + look-up table with appropriate interpolation between points
  + When gain values are dependent on many parameters and the number of chosen operating points are large 🡪 higher dimentional database 🡪 large storage size and computationally expensive to use look-up table 🡪
  + Use a polynomial fit of scheduling parameters 🡪 use look-up table for polynomial coefficients (smaller database, instead of storing Gain values, a polynomal for computating the gain values is stored)

### Fractional PID (Advanced)

* increase the DOF from 3 to 5
* Tuning parameters: 3 gains () + fraction of Integration and Differentiation order ()
* In normal PID, are one, integrate or differentiate one time.
* In fractional PID, they are not limited to one.
* Fractional Calculus

### Other Modifications

* Deadband: to reduce auturator wear
* Set-point change (reference generator: not to suddenly change the error when set-point is changed)
* Set-point weighting
  + Adjustable factors (0-1) in the setpoint of Proportional and Derivative terms.
  + Integral term use no weight, so the real error is gunranteed to be eliminated.
  + Can tune two additional parameters to improve the set-point response.

## PID Limitations

* Linear (Linearize the plant when tuning)
* Amplification of noise in derivative term
  + Solution: Low-pass filter, Low noise sensor, state observer(e.g. KF(less delay, need plant model, sensor fusion OK))
  + Problem: low-pass filter introduces delay, low-pass filter and derivative action can cancel out each other (just using PI may be simpler), Check phase margin or disc margin for stability.
* Not optimal in general
* Difficulties in case of
  + Process measurement dealy
  + Control action delay
  + In the presence of non-linearities
* Does not leverage the knowledge of process/plant/vehicle to be controlled

Cautions: The signs of the PID terms should be in accordance with the physical meaning of control variable (reverse control action, e.g. water level control (if error is negative(below the desired level), valve opening is positive))

## PID Tuning Methods

* Manual (when tune with real system, need to minimize how many time the trial and error is done, may be Bayesian optimization?)
* Ziegler-Nichols
* Tyreus Luyben
* Cohen-Coon
* Astrom-Hagglund
* Software Tool

Desired: Critically damped, balance between robustness and fast convergence

Stability (no bounded oscillation) is essential requirement.

Other two basic requirements:

* Regulation (disturbance rejection – staying at set-point)
* Tracking (respond to the set-point changes: rise time, settling time)
  + Some processes may not allow overshoot (due to the safety)
  + Other processes must minimize the energy when trying to reach a new set-point

Other requirements may conflict with one another.

## PID Robustness and Stability

* Gain and Phase Margins
* Disc Margin
* Nyquist Stability Criterion
* Pole Placement

## PID as Transfer Function and State Space Implementation

: Numerator order is higher than order of Denominator (not implementable)

: Filtered Derivative, N: Derivative Filter Coefficient

|  |  |
| --- | --- |
|  | **Code for PID Control (Class Implementation)** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **class** **PID\_Object(**object**):**  # Constructor  **def** \_\_init\_\_**(**self**,**  KP **=** 1.0**,** KI **=** 0.0**,** KD **=** 0.0**,**  u\_bounds **=** **(None,** **None)):**  self**.**KP **=** KP  self**.**KI **=** KI  self**.**KD **=** KD  self**.**u\_min**,** self**.**u\_max **=** u\_bounds  self**.**Integral **=** 0 # To keep the Intergral State  self**.**error\_last **=** 0 # Since there is error\_last at intial point    **def** \_\_call\_\_**(**self**,** error**,** dt**):**    self**.**Proportional **=** self**.**KP**\***error  self**.**Integral **+=** self**.**KI**\***error**\***dt  self**.**Integral **=** clamp**(**self**.**Integral**,**self**.**u\_min**,**self**.**u\_max**)**  self**.**Differential **=** self**.**KD**\*(**error**-**self**.**error\_last**)/**dt  u **=** self**.**Proportional **+** self**.**Integral **+** self**.**Differential  u **=** clamp**(**u**,**self**.**u\_min**,**self**.**u\_max**)**  self**.**error\_last **=** error  **return** u  # Helper Functions  # Clamp x between x\_min and x\_max  **def** clamp**(**x**,**x\_min**,**x\_max**):**  **if** x **<=** x\_min**:**  **return** x\_min  **elif** x **>=** x\_max**:**  **return** x\_max  **else:**  **return** x |

# Autopilot Improvements

* Wave Filtering
* Adaptation with Environment
* Reference and Disturbance Feedforward

# Sate Feedback Control

* Full Sate Feedback (FSF) (Linear Control)
* How the Feedback Gian “K” is chosen?
  + Pole Placement (Stability oriented)
    - Dyanamic System has its original poles
    - Eignevalues(A) = poles of the system 🡪 dicatates stability
    - We want to move these poles at where we want based on desired response
    - Characteristic equations are compared to compute required gain K
    - Single-Input system 🡪 Unique Gain K
    - Multiple-Input system 🡪 K is not unique, chooing the best gain K is not trivial
    - Pole Placement 🡺 Fancy Root Locus
      * Root Locus: only one gain K, can move only along Root-locus line
      * Pole Placement 🡪 Gian Matrix K, place anywhere
    - Moving poles to far left: large gain value 🡪 faster, larger acurator effort 🡪 consider physical limits
  + LQR , LQG = LQR + KF (Optimality oriented)
    - Gain K is choosed by solving ARE
* Feedback Linearization (Nonlinear Control)
* FSF is similar to PD Control. For a 2nd order system, the states are the postion/orientation and their derivative. So, it is similar to the PD with rate derivative term
  + For SISO system 🡪 silimar to PD with rate derivative term
  + For MIMO system 🡪 FSF is capable of coupling the sate and control variables
* Since it is FSF, all the states have to be meaurables or at least observable (e.g. KF)

# LQR Control

* Linear, Optimal Control
* State Feedback Control (Full State, Linear)
  + Optimal Feedback Gain K = solve ARE
  + If feedback states are estimated using KF 🡪 LQG
  + Equivalent to PD autopilot with yaw rate differential term (in place of error rate)
* Quadratic Cost Function
* LQR 🡪 derive state and control to zero (Regulator)
* For Tracking Control 🡪 Error Dynamics (ODE of Error) must be used instead of System Dynamics (State ODE)

|  |  |
| --- | --- |
| Linear System |  |
| Regulator | Drive |
| Performance Index  () | Or |
| Control Law  ( |  |
| Algebraic Riccati Equation (ARE)  (to solve for ) |  |

* Be careful! 🡺 Gains from Continuous ARE solution and Discrete ARE solution are quite different.
* Since Dynamics system is at disposal, LQR is usually used with Feedforward(FF) Terms (Reference FF and Disturbance FF)
  + Reference FF needs Reference Generator
* Since LQR is FSF and its behavior is similar to PD controller, for (unmeasureable) steady disturbance 🡪 Integral Effect is needed
  + LQR + I control
  + Integral-LQR (System matrix A, B are argumented with error integral term)
  + Reference scaling may be needed depending on the steady state error of the step response

## Tracking LQR

* Original LQR: regulate states/output to 🡪 zero
* For a arbitrary set-point 🡪 Instead of feeding back the steate, errors must be feedback

|  |  |
| --- | --- |
|  |  |
|  | For a set-point regulation,  For time-varying set-point (Tracking) Reference Generator |

## Integral LQR (I-LQR)

|  |  |
| --- | --- |
|  | is used to select the error from the error vector |
|  |  |
|  | One more sate () is added to the error vector |
|  | matrix has one more gain for Integral at (1,1) |

## LQR Limitations

* Pure LQR 🡪 No Integral Effect, so not good in case of Steady Disturbance
  + Distrubance FF should be used
* Compared to PID
  + LQR gains are automatically computed from system dyanmcics
    - No tuning on gains, but for weights
    - But, need to know (accurate) system dynamics
  + Coupled MIMO
  + Full Sate Feedback 🡪 Need to measure all the states
  + LQR: usually better performance than PID
    - PID is tuned by desired step response
    - LQR is tuned by choosing Weights depending on desired cost function design
* LQR does not consider acturator limits when solving for Gain
* Robustness 🡪 see LQR vs
  + It is well-known that a system with LQR controller has at least 60° phase margin and 6 dB gain margin.

## LQR Varients

* Pure LQR
* LQR + Integral, FF
* Integral LQR (can be thought as PID tuned by LQR)
* Finite Horizon LQR (Origianl LQR is infinite horizon control)
* **Discrete** LQR vs Continuous LQR

## LQR vs MPC

* Infinite Horizon vs Moving Short Horizon
* Unconstrained vs Constrained
* Offline optimization(solve ARE only one time) vs Online optimization (solve QP at every time step)
* MPC makes no assumption about linearity, it can handles hard constriants as well as migration of a nonlinear system away from its linearized operating point

# Synthesis

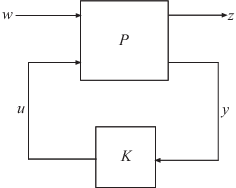
* Frequency Domain Design of MIMO Controller 🡪 Linear MIMO and Robust Control
* Not a controller itself, but a tuning method for predefined controller structure, or synthesizing a tuned controller(linear, may be higher order) based on optimization of norm, which is the cost function of design requirement. (MYO)
* 2 Types of Synthesis:
  1. Traditional Synthesis
     + Designs a full-order, centralized controller
     + Controller structure cannot be specified. Automatically generated.
     + Often results in a controller that has higher-order dynamics
     + Can be difficult to map to the specific real-world control architecture
     + Requires to express all design requirements in terms of a single weighted MIMO transfer function
  2. Structured Synthesis (Fixed-Structure Tuning)
     + Allow to specify the control architecture, including number and configuration of feedback loops.
     + Can specify the complexity, structure, and parameterization of each tunable component in the control system (e.g. PID, Gains K, Fixed-order TF/SS)
     + Can easily combine requirements on separate closed-loop TF
* Linear (Sub)Optimal Robust Control
* Readily applicable to problems involving multivariate systems with cross-coupling between channels (advantages over classical control techniques)
* Minimize the Energy Gain between reference change or disturbance and Cost function or weighted sum of errors.
* Minimize norm
* norm: the maximum singular value of the function over Hardy space (the name of the mathematical space over which the optimization takes place). (This can be interpreted as a maximum gain in any direction and at any frequency; for SISO systems, this is effectively the maximum magnitude of the frequency response.)
* Used to minimize the close-loop impact of a perturbation: depending on the problem formulation, the impact will either be measured in terms of stabilization or performance.

## Limitations

* Linear control
* Only optimal w.r.t the prescribed cost function and does not necessarily represent the best controller in terms of the the usual performance measures used to evaluate controllers such as settling time, energy expended, etc.
* Non-linear constrants such as saturation are generally not well-handled.

## LQR vs

* is based on the selection of weighting function whereas LQR is based on the selection of optimal FSF gain K.
  1. Weighting function 🡪 frequency dependent, depend on disturbance freq.
  2. Uncertainty Weight TF 🡪 satisfy nominal performance and robust performance criteria
  3. Output Weight TF 🡪 satisfy stability criteria
* No need to measure full states.
* Minimize the closeloop impact of a perturbation (energy gain of the system)
  1. Perturbation: disturbance, reference changes
  2. LQR is only optimized for reference changes
* LQR can be regarded as a special case of control with FSF as the controller.
* Unlike LQR, with FSF accounts for how disturbances affect the plant dynamics.
* System model include a term for disturbance/uncertainty
* So, cost function of also has that term and a corresponding weight ().
* That weight serves as a trade-off parameter between Performance(Optimality) and Robustness to disturbance/uncertainty
* If that weight is zero, with FSF is equivalent to LQR (assuming same cost function is used for both)
* Can’t achieve the highest Optimality and Robustness simultaneously.
* will ensure stability within designed uncertainty bounds (there will be trade-off in optimality), where as LQR may not. On the other hand, optimization in will not always give successful results since is linear control and plant with certain uncertainty cannot be stabilized by a linear controller. (MYO)
* LQR Gain matrix is diagonal matrix or row vector 🡪 No cross-coupling between states when computing . A general can have cross-coupling terms 🡪 the gain matrix is not limited to diagonal matrix.
* FSF 🡪 No state cross-coupling control terms
* For nonlinear plants, if the nonlinearities are within th bounds of uncertainty, will have not problem in controlling those nonlinear plants in term of stability. But other linear controls may have stability issues due to plant nonlinearities. (MYO)



|  |  |  |  |
| --- | --- | --- | --- |
|  | Traditional Synthesis | | |
| **MATLAB** | Optimal Controller | Full Control | Full Information |
| Remarks | A General Synthesis | Assumes the controller can directly affect both and | Assumes the controller has access to both and |
| Plant Equation |  | : Disturbance/uncertainty  : errors |  |
| Controller or  Gain | Controller = a SS Model  Same no. of states as  Input: (measurements)  Output: (controls) | Controller = Gain  : Inputs that affect  : Inputs that affect | Controller = Gain |

## norm

* minimizes maximum error while minimizes error function 2-norm

## Observer

* Kalman Filter 🡪 a special case of observer

## Robust Controller Synthesis

* Synthesis
* Synthesis 🡪 add parameter uncertainty in
* Loop-Shaping Synthesis
  + For simultaneously optimizing robust performance and robust stabilization
  + Apply classical loop-shaping concepts to the multivariable frequency response to get robust performance and then optimizes the response near the system bandwidth to achieve good robust stabilization.

## Robustness Assessment Methods

Add variations (gain and phase changes) in 🡪

* Plant output (input to controller)
* Controller output (input to plant)

Results 🡪

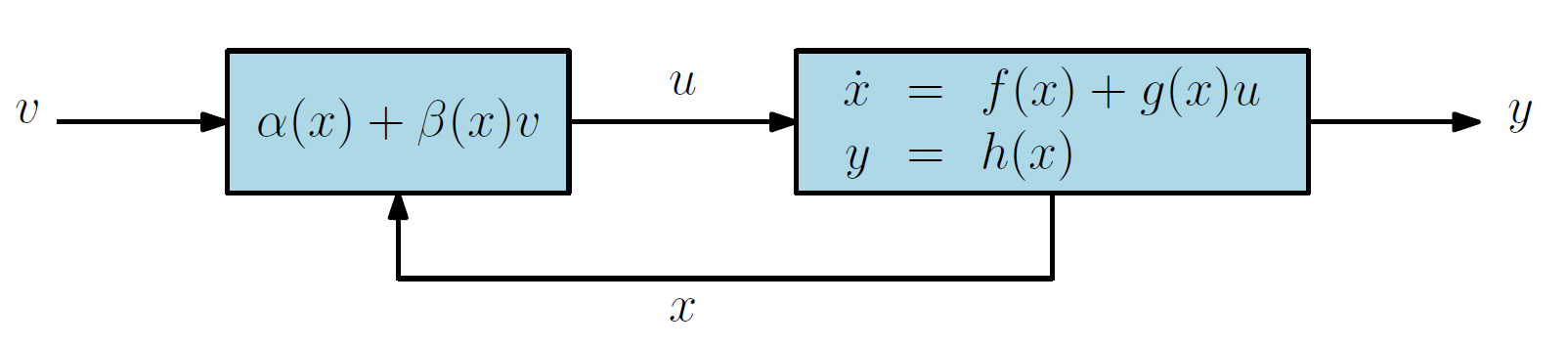
* Gain Margin
* Phase Margin
* Disc Margin (2D, gain and phase)

Add variations in model parameters

* Monte Carlo Approach (Random sampling)
* Deterministic Approach (MATLAB robstab command, similar to -analysis)

Combination of of above twos.

# Feedback Linearization





* A Nonlinear system is transformed into equivalent Linear system using “Nonlinear Control Mapping”
* Find relationship between (Original Control to Nonlinear System) and (Linear Control)
* Since and are functions of 🡪 Linearizing at current operating point (Linearize every time step based-on feedback state )
* Feedback Linearization 🡺 Finding and
* Then, use pole placement (FSF, PID), LQR, MPC to control the resulting equivalent linear system
* Usually used with FSF since state measurments are already available at FL
* Two Feedback Linearization
  + Full State(x) Feedback Linearization
  + Output(y) Feedback Linearization (Input-Output Linearization)
* Compared to Jacobian Linerization
  + FL is online: linearized at current operating point
  + Jacobian: Lineraized at chosen operating point once

## FL Process

* Control Mapping
* Obtaining Equivalent Linear System
* Design a linear controller

## Reference Models (LP Filters)

|  |  |  |
| --- | --- | --- |
|  | **A Single High-order ODE** | **Multiple 1st order ODEs** |
| Autopilot  (2nd Order) |  |  |
| Autopilot  (3rd Order) |  |  |
| Speed controller |  |  |

# Backstepping

* Nonlinear Control
* A Recursive Lyapunov-based scheme which achieve the **global stabilization**
* Aim: Global Stability (all operating points, not only local region)
* Can be used to design controller for plants with mismatched uncertainties (has Robustness, but not categorized as robust control)
* The idea is to design a controller recursively by considering some state variables as virtual controls and designing them for intermediate control laws ultimately reaching back to the actual control input.
* BS is a recursive methodology which designs the controller in a systematic stepwise manner by breaking the actual control into smaller parts called virtual control.
* It breaks down the problem into a number of lower-order control problems and a control law is recursively constructed.
* Conventional BS involves the use of a positive definite function to construct the virtual and actual controls.
* Adaptive BS (ABS): The control laws are recursively designed using input-ouput linearization theory.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

* Dynamics of is controlled by
* Dynamics of is controlled by and so on.
* Dynamics of final is controlled by actual
* At each step, sub-control laws are designed.
* At each step, Control Lyapunoc Function (CLF) is constructed to ensure stability.
* At the final step, the complete CLF is simply summation of the previous Lyapunov functions.
* Integrator Backstepping: the final CLF function is used to design the actual control input such that the resulting dynamics are always stable.

## Compared to Other Nonlinear Control Methods

* Does not require establishment of explicit timescale separation
* Compared to FL
  + More flexible way of dealing with nonlinearities
  + In FL, nonlinearities are canceled. In BS, their impact on the system can be analyzed using Lyapunov Functions. Useful nonlinearities can be retained
* Does not need Integeral Effect for steady disturbance(MYO)

## Example: Autopilot (Nonlinear 1st Order Nomoto Model)

|  |  |
| --- | --- |
|  |  |
|  |  |
| * 2nd Order ODE for * (here ) appears in 2 steps | Need only 2 Steps in Backstepping |
| By comparing, | Control Laws: FL + SF(Proportional) at each step   * : Nonlinaer mapping(FL) * : SF * : FL * : SF * reference Heading * rudder angle * : Control Gains |
|  | |

# Sliding Mode Control (SMC)

* Nonlinear Robust Control
  + A typical Robust Control consist of 2 parts:
    - A Nominal Part (Similar to Feedback control) and
    - A additional terms aimed at dealing with model uncertainty
  + SMC provides a systematic approach to the problem of maintaining **stability** and **consistent** **performance** in the face of modelling imprecision
* A Type of variable structure control (VSC)
  + A high-speed switched feedback control resulting in Sliding Mode
  + A discontinuous control changes the System Structure when the states reach the Intersection of Sets of Sliding Surface
  + A High frequency switching control
  + Origin: Relay control, bang-bang control
* Robust to Disturbance and Parameter Uncertainties
* Reduced order compensated dynamics
* **Matched** uncertainty doesn’t deteriorated the control quality (**bounded disturbance**)
* Require: High-Quality, High Bandwidth (fast) acturators
* Lyapunov Stability is applied

## Sliding Surface

* A high-speed switched feedback control resulting in Sliding Mode
* The gains in each feedback path switch between two values according to a rule that depends on the value of state at each instant
  + Usually, switch +ive and -ive depending on sign of error function (sliding variable)
* Switching control law 🡪
  + drives the nonlinear plant’s state trajectory onto a prespecified (user-chosen) surface in the state space and
  + maintain the plant’s state trajectory on this surface for subsequent time
  + That surface is called a switching surface (Sliding Surface, Sliding Manifold)
  + When the plant’s state trajectory is “above” the surface, a feedback path has one gain
  + A different gain if the trajectory drops “below” the surface
  + This surface defines the rule for proper switching
  + Ideally, once intercepted, the switched control maintains the palnt’s state trajectory on the surface for all subsequent time and the palnt’s state trajectory slides along this surface

## Lyapunov Function

* The most important task is: to design a switched control that will drive the plant state to the switching surface and maintain it on the surface upon interception.
* A Lyapunov approach is used to characterize this task
* **Lyapunov method is usually used to determine the stability properties of an equilibrium point without solving the state equation.**
* A generalized Lyapunov function (that characterizes the motion of the state trajectory to the sliding surface) is defined in terms of the surface e.g , is sliding surface or error function
* For each switched control structure, one chooses the gains so that the derivative of this Lyapunov function is negative definite, thus guaranteeing motion of the state trajectory on the surface
* After the proper design of the surface, a switched controller is constructed so that the tangent vectors of the state trajectory point towards the surface such that the state is driven to and maintained on the sliding surface.

## Higher order to 1st order system

* In the 1st order systems, the intuitive feedback control strategy “if the error is negative, push on the positive direction; if the error is positive, push on the negative direction” works.
* The same statement is not true for higher order system.
* In SMC, original tracking problem is transformed into the a simple 1st order stabilization problem in (sliding surface).
* The problem of tracking of n-dimentional vector can be replaced by a first order stabilization problem in
* ,

## Example 2nd order system (without )

|  |  |
| --- | --- |
| A 2nd Order System |  |
| Sliding Variable |  |
| Derivative of |  |
| By taking  Equivalent Control |  |
| Switching Control |  |
| Final Total Control |  |
| How to choose  Large enough such that: |  |
| Maximum allowable error/uncertainty 🡪 |  |
| 🡪 a positive constant to satisfy Sliding Condition |  |

## Example 2nd order system (with )

|  |  |
| --- | --- |
| A 2nd Order System |  |
| Sliding Variable |  |
| Derivative of |  |
| By taking  Equivalent Control |  |
| Switching Control |  |
| 🡪 Geometric Mean of and |  |
| Bounds for |  |
| Final Total Control |  |
| How to choose  Large enough such that: |  |
| Maximum allowable error/uncertainty 🡪 |  |
| 🡪 a positive constant to satisfy Sliding Condition |  |
| 🡪 Gain Margin  Bounded Multiplicative Uncertainty  ( bounds are rewritten with ) |  |

* A 2nd order system 🡪
* for 2nd order system 🡪 where , are kind of weights for errors, is chosen usually as one.
* If 🡪 will become zero. is chosen from this conditions
* is usually chosen as based on the sign of sliding surface or error function.
* For 🡪 it will drive to the sliding surface using high frequency switching
* The value of should be chosen as large enough such as:

## Ideal SMC:

* Control input drives the state variables to the sliding surface in finite time, and keep them on the surface there after in the presence of the bounded disturbance.
* Chattering Problem
  + Small Amplitude High Frequency zig-zag motion in state variables.
  + Caused by High Frequency Switching of Control
  + In ideal sliding mode: Switching Frequency 🡪 approach to and zig-zag motion Amplitude 🡪 tends to
  + Caused by imperfection in sign-function and discrete-time nature of computer simulation
* 2 Phases 🡪 Sliding Phase + Reaching Phase

## Uncertainties

* Modelling Inaccuracies
  + Structured(Parametric) uncertainties – Inaccurices on the terms actually included in the model
  + Unstructured uncertainties (Unmodelled Dynamics) – Inaccuracies on system order
* Disturbances
* Modeling inaccuracies can have strong adverse effects on nonlinear control systems.

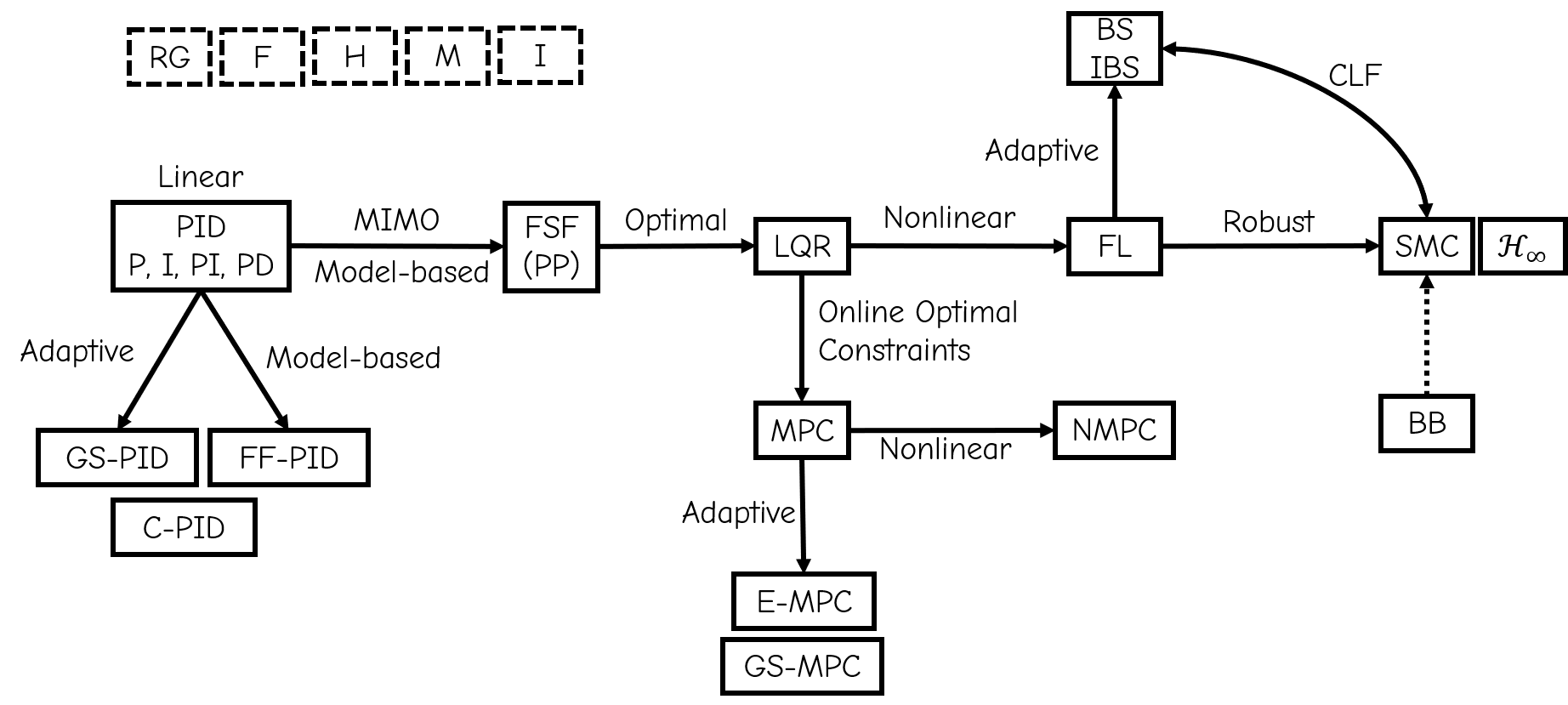
## SMC Autopilot (Example)

|  |  |
| --- | --- |
|  |  |
| Dynamic Model |  |
| Sliding Surface |  |
| Control Law |  |
|  | * By taking |
|  | When becomes zero (i.e. no error), Sliding Mode is reached and is deactivated, and only is used until next error appears. |
| Stability Proof | Control Lyapunov Function:  Finally,  If is chosen, for any value of , 🡺 stable |

## SMC Variants

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Names** | **Remarks** | **Smooth ?** |
| 1 | Ideal SMC | High Frequency Switching in | 🗴 |
| 2 | Quasi SMC | is replaced with Sigmoid function to smooth | 🌕 |
| 3 | Asymptotic SMC | Designed to get , then integrate 🡪 Smooth | 🗸 |
| 4 | Integral SMC |  | 🗸 |
| 5 | 2nd Order SMC | is not linear | 🗴 |
| 6 | Super Twisting SMC | 2nd order, Smooth , switching is under the integral | 🗸 |
| 7 | Nested SMC |  | 🗴 |
| 8 | Quasi Continuous Nested SMC |  | 🌕 |

# How to choose a control method?



|  |  |  |
| --- | --- | --- |
| **PID (P, I, PI, PD)** | **FSF (PP, LQR)** | **FL + #** |
| First Choice.  Work very well for Linear and Mildly Nonlinear SISO.  Use multiple for MIMO.  Use GS for Nonlinear.  For highly coupled nonlinear MIMO 🡪 may be “NO”. | For many DOFs, where multiple PIDs might be difficult to tune.  When all states are available.  When control efforts need to minimized too.  LQR: choose gains optimally based on a quadratic cost. | For nonlinear systems, where nonlinearities are known.  Cancel all nonlinearities  Combine with any linear control. |

|  |  |  |
| --- | --- | --- |
| **BS (IBS)** | **SMC** | **H∞** |
| For nonlinear system.  Gaurnatee to global stability.  Good nonlinearities remains.  In each step, can use any control method. | Nonlinear Robust control.  Bounded disturbances and uncertainties (know the bounds to choose gain). |  |

|  |  |  |
| --- | --- | --- |
| **MPC** | **EMPC, GS-MPC** | **NMPC** |
| Optimal control.  Constraints explicitly in controller.  For linear systems, but OK for some nonlinear systems with linearized model.  Cost function: only quadratic.  Constraints: only linear  Computaionally expensive | MPC for nonlinear systems. | System, Cost, Constraints: can be nonlinear.  Computaionally very expensive.  May have noisy control since NLP is limited. |

# Planning

## Motion Planning (Trajectory Planning, POSE and Derivatives)

## Path Planning (only POSE)

* Graph-based Methods
  + Search-based 🡪 A\*
  + Sampling-based 🡪 RRT, RRT\*

## zhm-real/PathPlanning

.

└── Search-based Planning

├── Breadth-First Searching (BFS)

├── Depth-First Searching (DFS)

├── Best-First Searching

├── Dijkstra's

├── A\*

├── Bidirectional A\*

├── Anytime Repairing A\*

├── Learning Real-time A\* (LRTA\*)

├── Real-time Adaptive A\* (RTAA\*)

├── Lifelong Planning A\* (LPA\*)

├── Dynamic A\* (D\*)

├── D\* Lite

├── Anytime D\*

└── Potential Field

└── Sampling-based Planning

├── RRT

├── RRT-Connect

├── Extended-RRT

├── Dynamic-RRT

├── RRT\*

├── Informed RRT\*

├── RRT\* Smart

├── Anytime RRT\*

├── Closed-Loop RRT\*

├── Spline-RRT\*

├── LQR-RRT\*

├── Fast Marching Trees (FMT\*)

└── Batch Informed Trees (BIT\*)

## zhm-real/MotionPlanning (Car)

* Hybrid A\* Planner
* Frenet Optimal Trajectory
* Hierarchical Optimization-Based Collision Avoidance (H-OBCA)

# A\* Path Planning

* A\* 🡨 Search-based 🡨 Graph-based
* Discretize the Occupancy Map into cells
* Define the cost for moving between cell

## Limitations

* Need fine resolution for good optimal path
* High-dimensional space, large area(space), fine resolution 🡪 high computational cost 🡪 use smapling-based methods (e.g. RRT\*)

## A\* Pathfinding (Sebastian Lague)

F cost = G cost + H cost

F cost = Total cost

G cost = cost(distance) from the starting node

H cost = Heuristic cost: cost(distance) from the end node, (defined by a cost function. Eg. Straight-line distance)

**Heuristic:** A heuristic technique, or a heuristic, is any approach to problem solving or self-discovery that employs **a practical method** that is **not guaranteed to be optimal, perfect, or rational**, but is nevertheless **sufficient for reaching an immediate, short-term goal or approximation**. (Wikipedia)

### Sudo Code

|  |
| --- |
| **OPEN** // the list(set) of nodes to be evaluated  **ClOSED** // the list(set) of nodes already evaluated (their neighbours)  add the start node to **OPEN**  loop  **current** = node in **OPEN** with lowest f\_cost  remove **current** from **OPEN**  add **current** to **CLOSED**  if **current** is target node // path found  return  foreach **neighbour** of **current** node  if **neighbour** is not traversable(obstacle) or **neighbour** is in **CLOSED**  skip to the next **neighbour**    if new path to **neighbour** is shorter or **neighbour** is not in **OPEN**  // shorter ==> update with new smaller f\_cost and its parent is also updated  // not in OPEN ==> set the f\_cost for first time and add that to OPEN  set f\_cost of **neighbour**  set parent of **neighbour** to **current**  if **neighbour** is not in **OPEN**  add **neighbour** to **OPEN** |

# Ship Path Planning for Collision Avoidance (Research on algorithms for autonomous navigation of ships, Agnieszka Lazarowska, July 2019 Publisned, December 18 Received)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Year** | **Name** | **Full Name** | **Run Time** | **Comments** |  |
| 2017 | FMM | Fast Marching Method | < 1 s | Deterministic | FMM |
| 2017 | VD | Voronoi Diagram | < 0.6 s | Graph-Search |  |
| 2016 | APF | Artificial Potential Field | - | Deterministic | PF |
| 2016 | DG | Differential Game | A few s |  |  |
| 2015 | BPF | Baterial Potential Field | < 10 s | Deterministic | PF |
| 2015 | EEA\* | Energy Efficient A\* | ms | Graph-Search | A\* |
| 2015 | FMM | Fast Marching Method | 0.1 s | Deterministic | FMM |
| 2014 | ANN | Artificial Neural Network | - |  |  |
| 2014 | FL | Fuzzy Logic | - |  |  |
| 2013 | CPP | Cooperative Path Planning | 7 s | Deterministic |  |
| 2012 | PSO | Partical Swarm Optimization | - | Stochastic | Swarm |
| 2012 | EA | Evolutionary Algorithm | < 60 s | Stochastic | EA |
| 2012 | A\* | A\* | - | Graph-Search | A\* |
| 2011 | PFM | Potential Field Method | - | Deterministic | PF |
| 2010 | EA | Evolutionary Algorithm | 200-800 s | Stochastic | EA |
|  |  |  |  |  |  |
|  | **Proposed in that paper** | | | |  |
|  | ACO | Ant Colony Optimization | 0.2-1.03 s | Heuristic, Swarm Intelligent | Swarm |
|  | TBA | Trajectory Base Algorithm | 10-30 s | Deterministic, Search a base trajectory |  |

# Path Planning Method Catagories

* Graph-based Methods
  + Search-based
    - Dijkstra (A\* without h cost)
    - **A\*** and variats
    - D\* Dynamic D\*(incremental heuristic search)
  + Sampling-based (RRT and variats)
    - RRT (basic)
    - **RRT\*** (not necessarily connect with nearest node, more optimal)
    - Informed RRT\* (use heuristic cost like Dijkstra is improved to A\*)
  + Any Angle Path Planning
    - A\*-based (Theta\*, Filed D\*)
    - RRT-based (RRT methods are Any Angle Path Planning Method)
  + Incremental heuristic search (replan fast using the experience of previous planned path)
    - Fringe Saving A\*
    - Generalized Adaptive A\*
    - Lifeling Planning A\*, D\*, D\* Lite
* Road Map Planning (Sampling-based, also use graph-search)
  + Voronoi Road Map Planning
  + Visibility Road Map Planning
  + Probabilistic Road Map Planning
* Potential Field Methods
  + **APF – Artificial PF**
  + BPF – Baterial PF
* **Fast Marching Method**
* Swarn Intelligent
  + PSO
  + ACO
* **Evolutionary Algorithms**
* Trajectory Units
* Optimal Control
  + **Trajectory Optimization** (Full Horizon, Offline)
  + (N)MPC (Receeding Horizon, Online)
    - Lookup Table (MPC generated trajectories)
* Miscellaneous
  + ANN
  + RL (Reward-based)
  + Fuzzy Logic
  + Dubin (Straightline + Arc Path)
  + Reed-Shepp (Dubin with Reverse capability)
  + Waypoint connecting (Spline interpolation, Dubin, WOP(ship tracking autopilot))
  + LQR-based
  + State Lattic Planning, (Trajectories generated with MPC) Sampling methods:
    - Uniform Polar Sampling
    - Biased Polar Sampling
    - Lane Sampling

# Filters (Digital)

## Transfer Function(TF) to State Space(SS) Representation

|  |  |
| --- | --- |
| Transfer Function  (**s**/Lapace Domain) |  |
| State Space  (ODE + Output Equation) |  |

e.g.

|  |  |
| --- | --- |
| 2nd Order Transfer Function |  |
| Equivalent State Space Representation |  |

## Filters

|  |  |  |
| --- | --- | --- |
| Filters | Transfer Function | State Space Representation |
| Lowpass |  |  |
| 1st Order |  |  |
| 2nd Order |  |  |
| 3rd Order | 3rd Order = 2nd order x 1st order |  |
| 4th Order | 4th Order = 2nd Order(zeta1) x 2nd Order(zeta2) |  |
| Notch(2nd) | : 0 to 1 (0 = highest magnitude of notch , 1 = no filtering) |  |
| Highpass | : 0 to 1 (Strenght of filter, 0 = no filtering, 1 = highest) |  |
| Derivative | : not implementable due to larger order at numerator  cascaded with above 1st order TF to approximate Derivative |  |

# System Identification

## Model Classification

* Online and Offline
* SISO, SIMO, MISO, MIMO
* Linear(TF, SS) and Non-linear
* Gray Box, Black Box and White Box
* ODE or Input-Output
* Simulation and Control-oriented (High Fidelity vs Low Fidelity)

## Preprocessing

* Remove “Means”, “Trends”
* Filtering (Remove Measurement Noise)
* Resample
* Transform (Linear, Log)
* Combine, Split

## Models

* Linear
  + **Transfer Function (TF)**
  + **State Space (SS)**
  + Process (Low order TF with static Gain, Time Constant, Input/Output Delay)
  + Input-Output Polynomial (ARX, ARIX, ARMAX, ARIMAX, BJ(Box-Jenkins), OE(Output-Error))
  + Spectral (Frequency Response Model obtained using Spectral Analysis)
  + Correlation (Impulse Response Model obtained using Correlation Analysis)
  + **Grey-Box** (**ODE** Coefficients, SS Model Coefficients)
  + LSTM (How about LSTM for Non-linear, not better than ARX Models?)
* Non-linear
  + **Nonlinear-ARX** (Non-linear Estimators:)
    - **Neural Netwrok (NN)**
    - Sigmoid Network
    - Wavelet Network
    - Binary-Tree
    - Linear Non-linear Estimator (piece-wise linear?)
    - Custom Netwrok
  + Hammerstein-Wiener
  + **Grey-Box (Nonlinear ODE)**
* Timeseries Analysis (Not external input, could be used to predict other vehicles’ trajectory)
  + AR, ARI, ARMA, ARIMA
  + Nonlinear ARs

## Online SI (especially for Time-varying System)

* Filters (**KF, EKF, UKF, PF**, etc.)
* ARX, ARMAX, Recursive Least-Square

## Terminology

|  |  |  |
| --- | --- | --- |
| **TF** | Transfer Function | Differential Equation Model |
| **SS** | State Space |
| **OE** | Output-Error | Input-Output Polynomial |
| **BJ** | Box-Jenkins |
| **ODE** | Ordinary Differential Equation | Differential Equation Model |
| **AR** | Auto-Regressive | Input-Output Polynomial  AR without External Inputs  Linear Time Series Analysis |
| **ARI** | Auto-Regressive Integrated |
| **ARMA** | Auto-Regressive Moving-Average |
| **ARIMA** | Auto-Regressive Integrated Moving-Average |
| **ARX** | Auto-Regressive Exogenous | Input-Output Polynomial  AR with External Inputs  Linear Dynamic Models  Can also be used for Online Models |
| **ARIX** | Auto-Regressive Integrated Exogenous |
| **ARMAX** | Auto-Regressive Moving-Average Exogenous |
| **ARIMAX** | Auto-Regressive Integrated Moving-Average Exogenous |
| **N-ARX** | Non-linear ARX | Various Non-linear estimators (NN,Sigmoid,Wavelet) are introduced in the ARX models |
| **LSTM** | Long-Short Term Memory | An RNN Model  Could be non-linaer depending on Activation function |
| **KF** | Kalman Filter | State Observers (Estimators)  Sensror Fusion Algorithms  Online Dynamic Models (Time-varying) |
| **EKF** | Extented Kalman Filter |
| **UKF** | Unscented Kalman Filter |
| **PF** | Particle Filter |
|  |  |  |
| **SISO** | Single-Input Single-Output |  |
| **SIMO** | Single-Input Multiple-Output |
| **MISO** | Multiple-Input Single-Output |
| **MIMO** | Multiple-Input Multiple-Output |

## Model Structure (ODE, SS) of Various Vechicles

* Selected Model Types
  + Dynamics (DE)
    - TF
    - SS
    - ODE (Linear and Non-linear)
  + Input-Output Relationship
    - ARX (Linear and Non-linear, Polynomials)
* Marine: Fossen(ODE) and MMG(ODE, usually non-linear)

## Software Tools

* MATLAB System Identification Toolbox
  + TF, SS, ODE (Linear and Non-linear)
* PySINDy
* SciPy Optimization

## Linearization

* Step 1: Choose an operating point (desired equilibrium state “x”)
* Step 2: Trim the system (find the correspoinding “u” for chosen “x”)
* Step 3: Linearize using truncated Tayler series (1st order, Jacobian Linearization, assume function is differentiable)
* Simulink Linearization:
  + Not linearize the system as a whole
  + Each individual block or function has its own linearized form (i.e. analytical Jacobian or assigned Linear form)
  + If there is no predefined Jacobian, use numerical perturbation
  + If the function is not differentiable at operating point (e.g. discontinuous), use system knowledge to manually assign Jacobian of that function
  + Then, to obtain the overall linearized system, all the blocks are combined based-on their position in the block diagram
* Another way to obtain Linear Model is: System Identification with Linear Models
  + Generate fitting data using nonlinear model (make sure to include critical operating coditions)
  + Fit them to a linear model (e.g. TF, SS)

# Software Packages

* ArduPilot