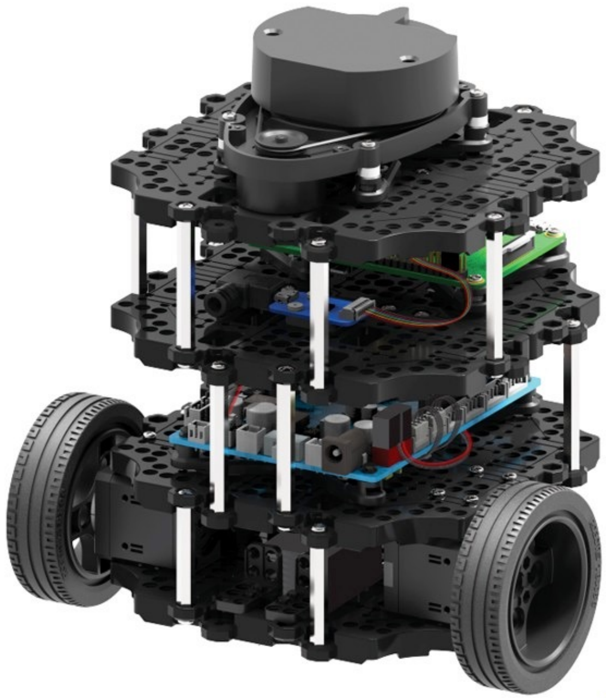
Description of Project 1

PID Control in ROS

EE3305/ME3243 Robotic System Design

A/Prof. Prahlad Vadakkepat

Dr. Andi Sudjana Putra

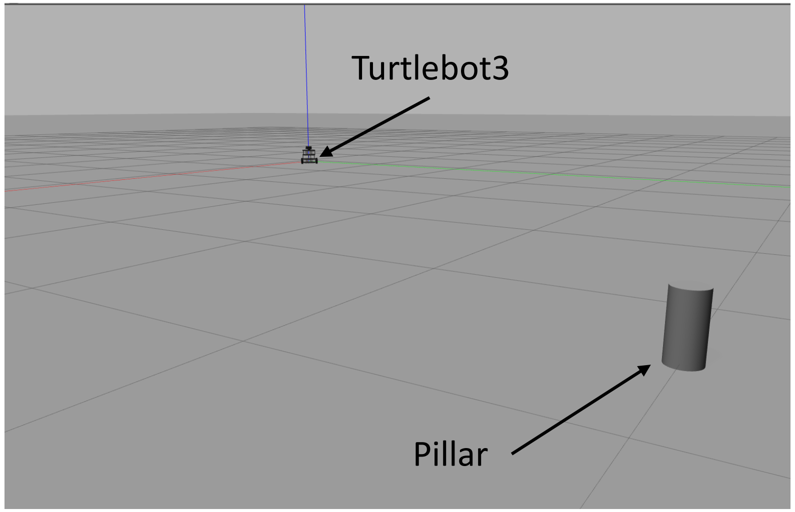


Project 1: Summary

You will be controlling a Turtlebot3 in a ROS environment.

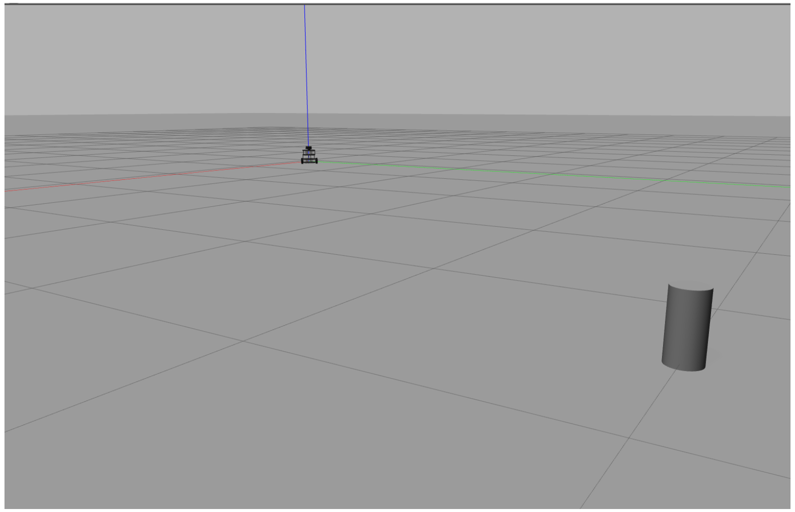
Turtlebot3 will start from an initial position.

The target pillar is at a predetermined positioned not previously known to the robot.



Project 1: Task

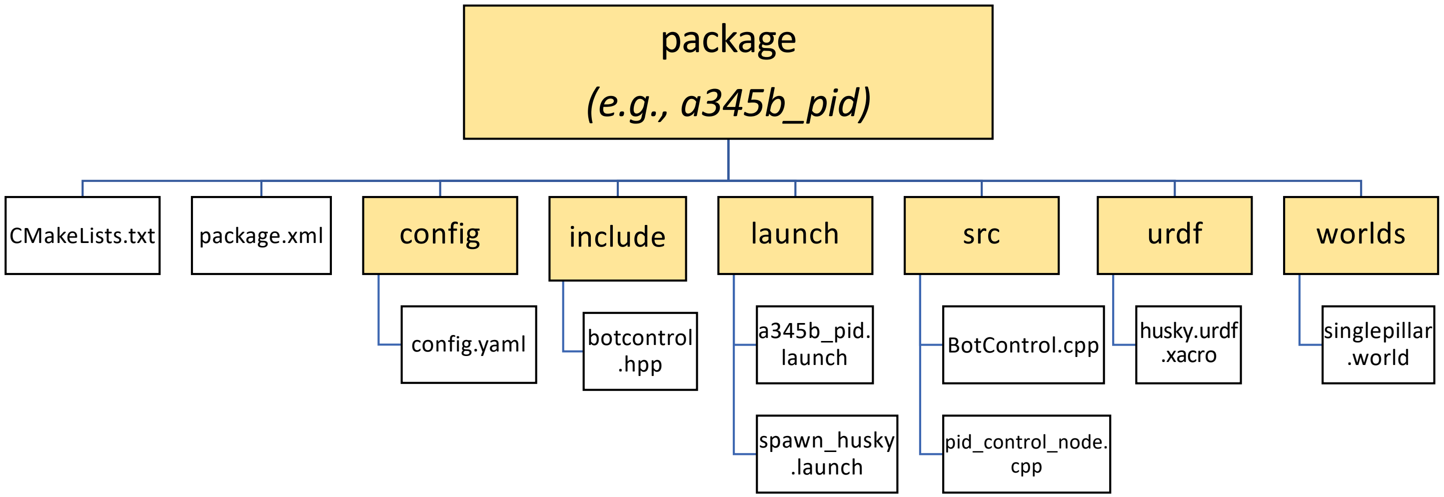
- Design a PID control to drive the Turtlebot3 autonomously towards the pillar and to stop at a predetermined distance.
- Define the performance criteria. Example:
    - Steady state error of …
    - Overshoot of …
    - Settling time of …
    - …
- Simulate the motion. Analyse how the motion is (steady state error, overshoot, etc.). Consider the constraints that physical systems have.



Project 1: Structure of PID Control

- Assumption: the motoring control (forward-reverse motion) is completely decoupled from steering control (left-right turning motion)
- Consequence of the assumption: design two PID controls, one for motoring control and another one for steering control
- Suitable for two (or more) decoupled systems, although they may be housed in an embodiment
- Example: a platform with independent steering and forward-backward movement

File structure (example: a345b_pid)



Save your `package`.

Report

1) Initial conditions.
    a) Show the initial location of the pillar and the Turtlebot3 of your setting. Use the view that best show their locations.
    b) Calculate the initial distance and orientation of the pole with respect to the Turtlebot3.
2) Implementation of PID control. Discuss how the PID control is implemented in ROS with reference to your code.
    a) Describe how you (1) define the integral term, (2) define the derivative term and (3) define the PID control term.
    b) Describe the purpose of the code to (1) regularize the angular error (error_angle) and (2) limit the angular control signal (trans_angle).

Report

3. Tuning of PID.
    a) Discuss the tuning process, e.g., which gain is determined first, which gain is determined second, how it is determined and so on.
    b) Discuss how you would characterise the PID control (P, PI, PD or PID). Discuss the merits, demerits, and other points that you want to highlight about your design.
4. Performance of PID control.
    a) Attach the plots of errors vs time (both linear and angular errors) that represents your best design.
    b) Analyse the performance, e.g., overshoot, steady state error and settling time.
5. Conclusions and key learning points.

What is a good design?

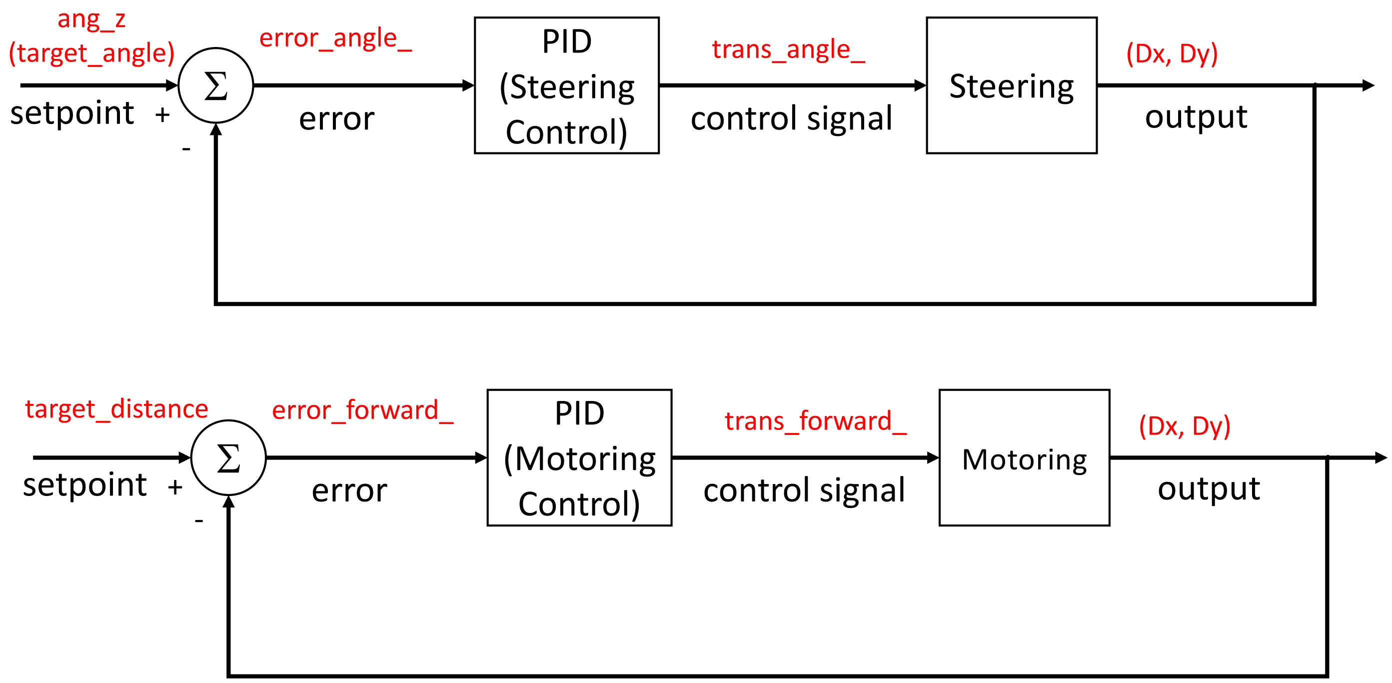
$3 + 5 + 7 = ?$

$? + ? + ? = 15$

Description of ROS Codes

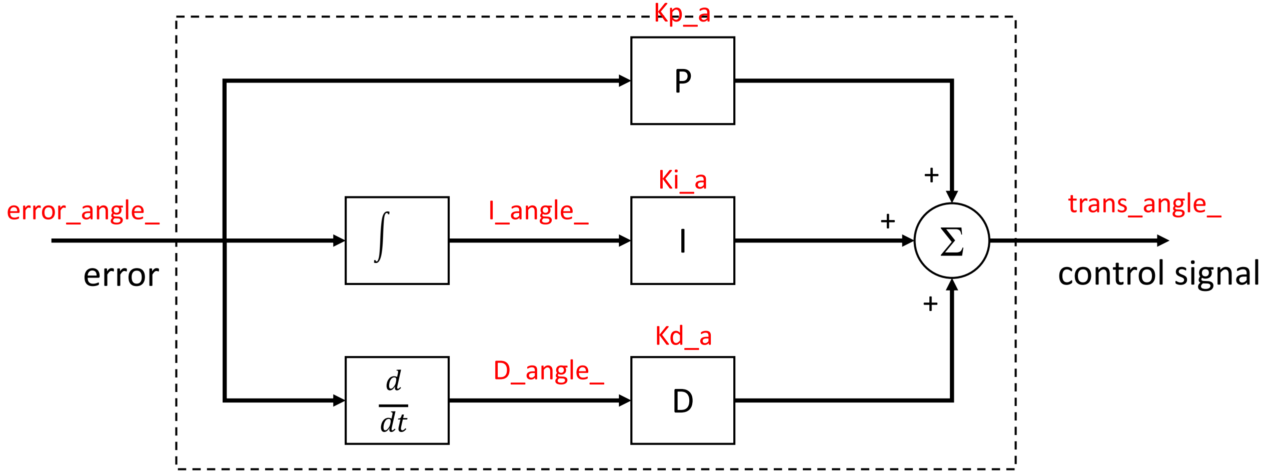
Control Structure and Its Variables

(Project 1)



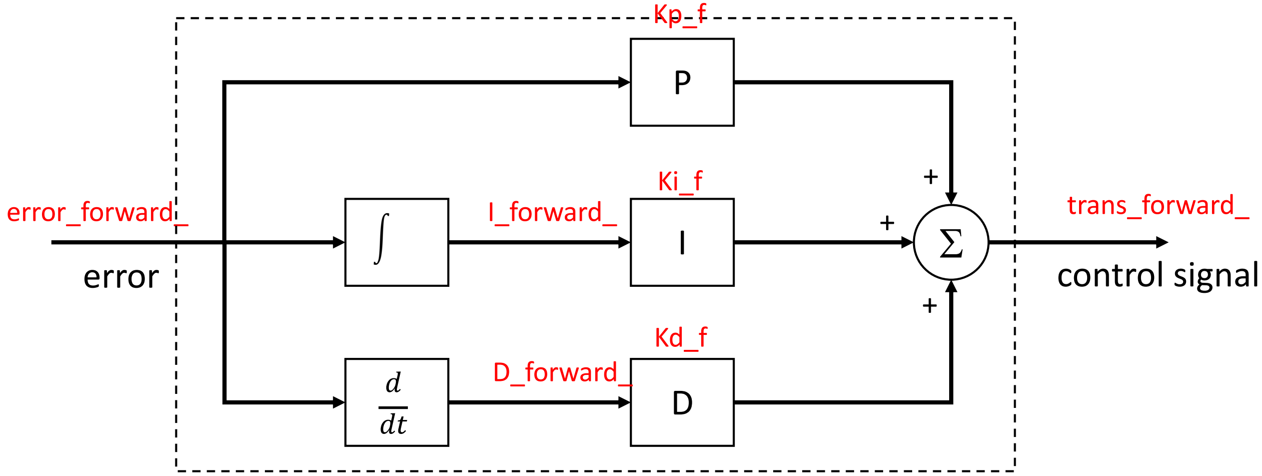
Steering Control

(Project 1)

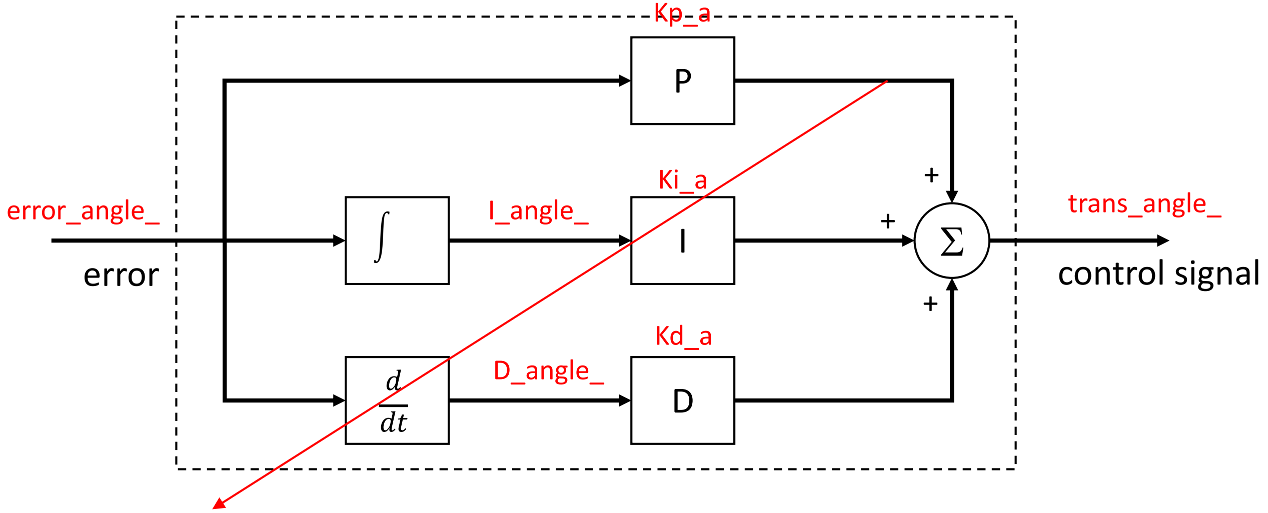


Motoring Control

(Project 1)



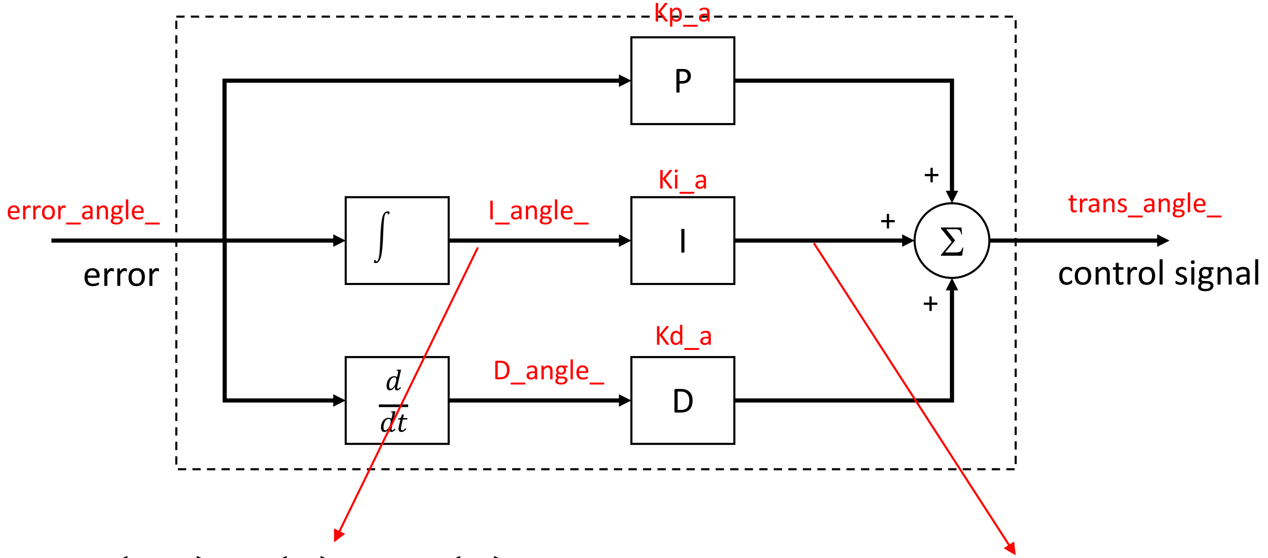
Proportional (Steering)



ROS Implementation:
Kp_a * error_angle_

Can you derive the ROS Implementation for motoring?

Integral (Steering)



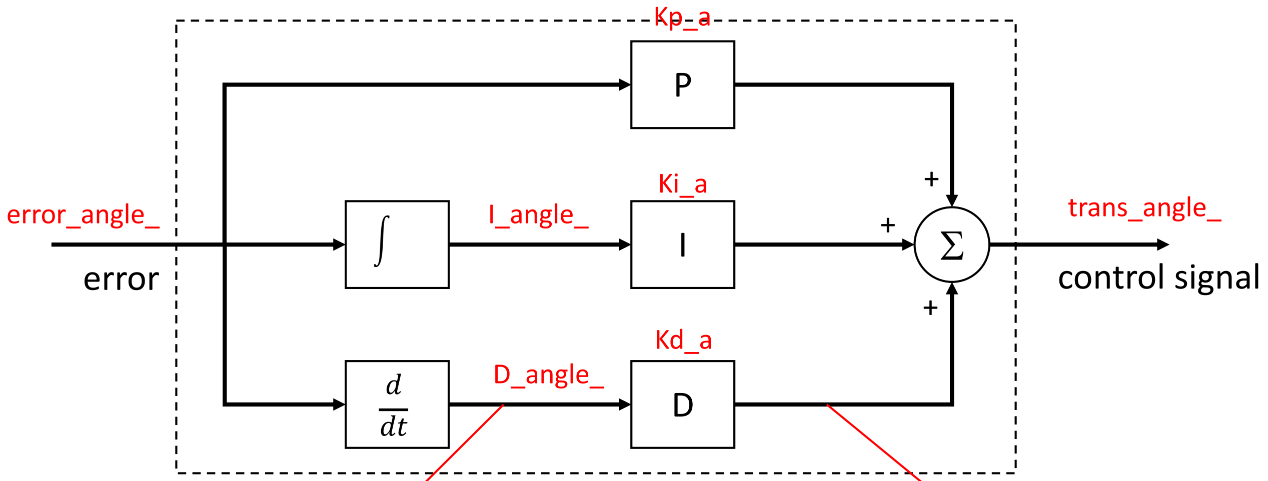
From $\mathrm{I}(\mathrm{t}_{k+1}) = \mathrm{I}(\mathrm{t}_k) + \mathrm{h} \cdot \mathrm{e}(\mathrm{t}_k)$

ROS Implementation:
I_angle_ = I_angle_ + dt * error_angle_
(OR) I_angle_ += dt * error_angle

ROS Implementation:
Ki_a * I_angle_

Can you derive the ROS Implementation for motoring?

Derivative (Steering)



From $D(t_k) = \frac{e(t_k) - e(t_{k-1})}{h}$

ROS Implementation:
D_angle_ = (error_angle_ - error_angle_prev_) * dt

ROS Implementation:
Kd_a * D_angle_

Can you derive the ROS Implementation for motoring?

Launch the simulation

- Setup the singlepillar.world
- Call the Turtlebot3 and spawn it at the origin
- Launch PID control

Enable control of robot

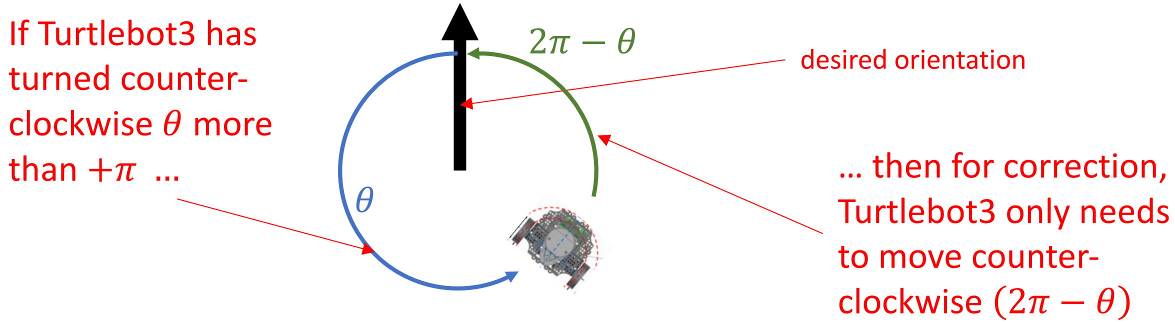
- In botcontrol.hpp
  - **packages**: sensor_msgs, nav_msgs, geometry_msgs, std_msgs, gazebo_msgs
  - **private variables**:
    - ROS topics of the simulation
    - PID-related: error_forward_, error_angle_, error_forward_prev_, error_angle_prev_, I_forward_, I_angle_, D_forward_, D_angle_, trans_forward_, trans_angle_
    - Turtlebot3-related: Dx, Dy
  - **public variables**: Kp_f, Ki_f, Kd_f, Kp_a, Ki_a, Kd_a, target_distance, target_angle, pillar_x, pillar_y, dt

Setup PID control

- In BotControl.cpp and pid_control_node.cpp
  - ROS topics of the simulation are declared
  - PID-related variables are initialised: error_forward_, error_angle_, error_forward_prev_, error_angle_prev_, I_forward_, I_angle_, D_forward_, D_angle_,
  - Turtlebot3-related variables are defined: scan_range_, scan_angle_
  - **public variables**: Kp_f, Ki_f, Kd_f, Kp_a, Ki_a, Kd_a, target_distance, target_angle, dt

PID Algorithm (BotControl.cpp)

- Regularise error_angle_ within $[-\pi, +\pi]$



if(error_angle_ > $+\pi$) error_angle_ -= 2π

Can you derive for the opposite angle?

Parameters to setup

(Project 1)

- Obtained from public variables in BotControl.hpp
- Defined in config.yaml

Kp_f:
Ki_f:
Kd_f:
Kp_a:
Ki_a:
Kd_a:
pillar_x:
pillar_y:
target_distance:
target_angle:
dt: 0.1 (assume this is the computation rate)

to tune these parameters

according to last digits of matriculation number

CMakeLists.txt

- find_package
- catkin_package
- include
- add_executable
  - src/pid_control_node.cpp
  - src/BotControl.cpp
- target_link_LIBRARIES

roscpp, sensor_msgs, nav_msgs, geometry_msgs, std_msgs, gazebo_msgs

form one node (PID control)

package.xml

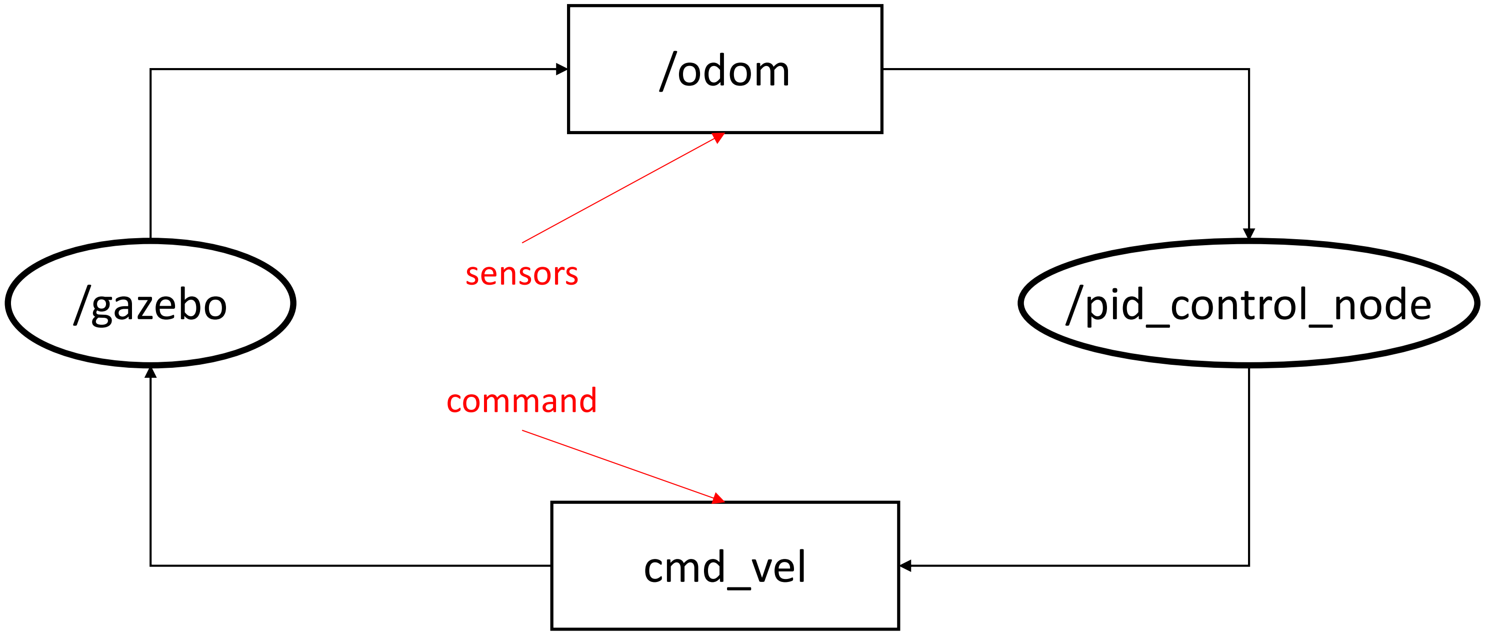
- <buildtool_depend>catkin</buildtool_depend>
- <build_depend>…</build_depend>
- <build_export_depend>…</build_export_depend>
- <exec_depend>…</exec_depend>

roscpp, sensor_msgs,
nav_msgs, geometry_msgs,
std_msgs, gazebo_msgs

To affect changes in ROS

- Build the package
- Source the workspace
- Launch

Exploring ROS Simulation Structure



Thank you.