TEAM ABHIYAAN

Mechantronics Application

Sub-vertical: Mechanical

By Teja, AE25B015

1.Answer to question 1:

Part A:

Why open loop is chosen over closed loop?

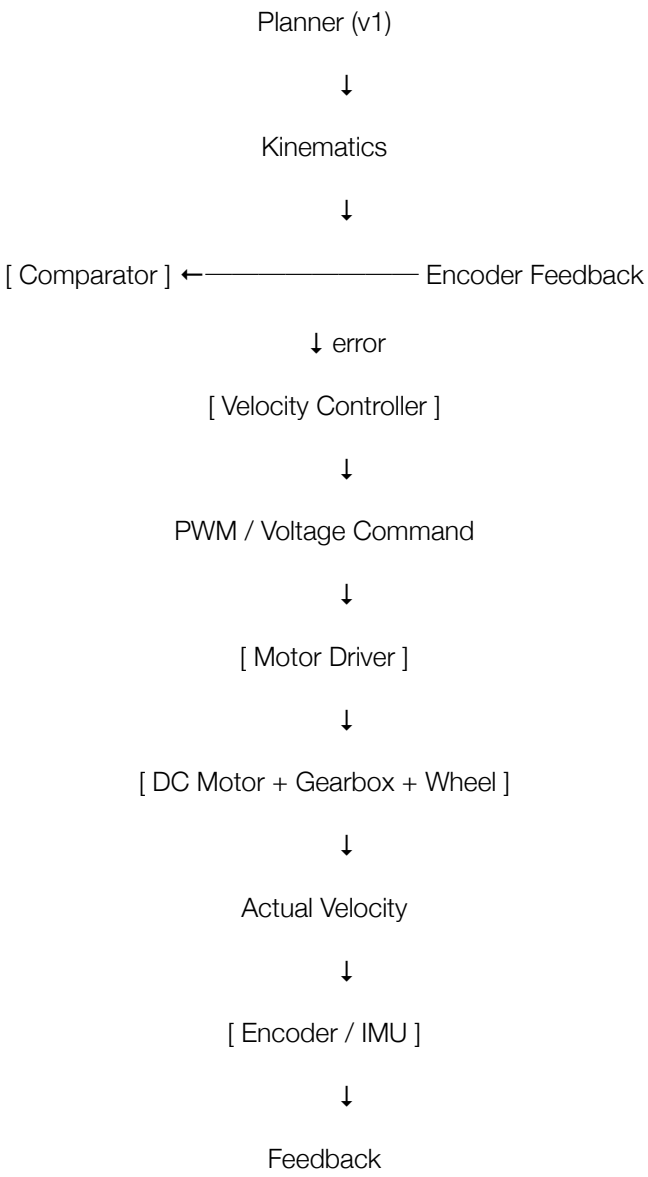
An **open-loop system** acts **without using feedback**. It does **not measure the output**, and it does **not correct errors automatically**. Examples include Electric toaster, Washing machine timer, Traffic light timer etc. If you set 2 minutes, it runs for 2 minutes — it never checks the result. Open loop is preffered when:

1. Output is **not critical**
2. System behavior is **predictable**
3. **Human supervision exists**
4. Cost must be **low**

But when accuracy is required we need a closed loop.

So now we have an autonomous mobile robot, which requires a good controller for a smooth ride. The robot receives commands from a high level planner to set the setpoint for the robot's velocity say (V1). Now since the problem is with control strategy and how to maintain the velocity 'set' despite of all disturbances like friction delay etc., here's my proper solution for it.

- So, we have a controller in every motor of the wheel.
- Encoders or IMUs give the feedback(sensors), and its a closed loop control system btw.
- I am taking the example of a 'differential drive robot', meaning a robot having two wheels and a castor..
- The two wheels have two motors.
- So now here we have a block flow diagram of how the actual signal flow and control:



- Let me explain this one by one.
- Planner gives the desired velocity v1, then kinematics, does the physics part as to how much angular velocity is 'set';let's say it w1.
- Encoder gives w2 which is the current anghular velocity at which its rotating.
- Comparator is the which takes the both values and calculates the error $e(t) = SP - PV$. SP is SET POINT

which is the set velocity, whereas PV is PROCESS VARIABLE which we get from the sensor.

- Then comes the actual controller which **tries to set the velocity by giving some output value to acuators.** This is called **MV - Manipulated Variable.**
- **MV** depends on which control stategy we use.
- Before explaining how velocity controller (a programmed one) converts to the physical motion and counters disturbances, I will briefly explain my control stategy and how it works for this robot.
- I have to learn abt other advanced control systems but I strongly believe for a differential drive robot a normal Cascaded PID (Proportial, Integral, Derivative) is a strong strategy. Here are some reasons:

  1. In a On-Off Control output is either max or zero which does not give smooth motion to the robot.
  2. Single PID is good but the D term fluctuates. So the motor jitters. PWM (Pulse Width Modulation jumps suddenly).

- **Cascaded PID means using multiple PID controllers arranged in layers (loops), where one controller's output becomes the input (setpoint) of the next controller.**

  Instead of controlling everything with one PID, we **split the problem into fast and slow parts.**

  Outer loop: controls the main goal (like velocity).
  Inner loop: controls a faster variable (like torque/ current).

- Now lets some to handling noise and disturbances. THe following describes how the disturces are countered:

Problem	Solution
Encoder noise	Low-pass filter

Derivative noise	Avoid large D
Friction	Integral action
Load changes	Fast torque loop
Delay	High-rate inner loop
Saturation	Anti-windup
Battery drop	Current control

->Assumptions

- Wheels do not slip excessively.
- Encoder measurements are reliable.
- Motor dynamics are approximately linear.
- Sampling rate is fast (>500 Hz inner loop).
- Battery voltage varies slowly.

->TRADE -OFFS:

Accuracy vs Complexity

- Cascaded PID is chosen to improve accuracy.
- Slightly more complex than single PID, but gives better performance and stability.

Noise vs Response Speed

- Derivative term improves response.
- Filtering is used to reduce noise amplification.

Cost vs Performance

- Using both **encoders and IMU** increases cost.
- But it significantly improves feedback quality and robustness.

Speed vs Stability

- Very high gains make the system fast but unstable.

- Moderate gains balance quick response with safe, smooth motion.

Model Dependence vs Robustness

- PID does not rely heavily on an exact system model.
- It is more robust and easier to implement than advanced methods like MPC.

- Now lets come to mathematics part of the robot's control system:

  Velocity loop:

$$e_v = \omega_1 - \omega_2$$

$$T_d = K_{pv} e_v + K_{iv} \int e_v \, dt + K_{dv} \frac{de_v}{dt}$$

  T_d is desired motor torque, $K_{pv}\ K_{iv}\ K_{dv}$ are tuning parameters for velocity loop.

  Inner Torque loop:

$$e_i = T_d - T_m$$

$$u = K_{pi} e_i + K_{ii} \int e_i \, dt + K_{di} \frac{de_i}{dt}$$

  As said before, output of outer loop becomes input of inner loop which calculates the **u** which we call as the PWM duty cycle.

- Now let's have brief explaination of PWM (Pulse Width Modulation) working.

  PWM (Pulse Width Modulation) is a way to control **power delivered to a motor (or any load)** by rapidly switching the voltage ON and OFF, instead of changing the voltage level itself.

  We don't send "half voltage".
  We send **full voltage, but for less time**. So the motor sees an **average voltage**.

- PWM controls power by rapidly switching a signal ON and OFF, and varying how long it stays ON during each cycle. This ON time relative to the total period is

called the **duty cycle**, and it determines how much power is delivered to the actuator. The duty cycle is defined as

$$\textbf{Duty Cycle} = \frac{\textbf{ON Time}}{\textbf{Total Time}} \times 100\,\%$$

A duty cycle of 0% means the signal is always OFF and no power is supplied, while 25% means the signal is ON for only a quarter of the time, giving low power. At 50%, the signal is ON half the time, producing medium power, and at 75% it provides high power. A duty cycle of 100% means the signal is always ON and the actuator receives full power. In motor control, changing the duty cycle directly changes the motor voltage and therefore controls speed or torque smoothly and efficiently.

- So the actuator which is the DC motor in Diff. Drive robot gets 'power' commands from the PWM and controls the 'Left and Right' wheels' speed accordingly.

—>Coming to the hardware part.

- To run a cascaded PID motion control loop on real hardware for an autonomous mobile robot, the selected platform must execute control algorithms reliably and in real time while interfacing with sensors and actuators.
- Among microcontrollers, platforms such as Arduino Uno/ Nano, Raspberry Pi Pico (RP2040), ESP32, STM32, and Teensy 4.x were evaluated.
- Arduino boards are easy to use and provide basic PWM and GPIO, but they are limited in processing power and memory.
- The Raspberry Pi Pico offers dual-core real-time performance with plenty of GPIO and is suitable for small robots.
- ESP32 provides wireless connectivity along with control capability, making it useful when communication is required.
- However, STM32 microcontrollers (such as the F4/F7 series) stand out because they provide industrial-grade peripherals, including high-resolution timers,

PWM channels, ADCs, and dedicated encoder interfaces, making them ideal for precise motor control.

- Single-board computers such as Raspberry Pi, BeagleBone Black, and Jetson Nano/Orin were also considered. Raspberry Pi boards run Linux and support high-level frameworks like ROS, vision processing, and navigation, but they are not deterministic for real-time motor control because the operating system introduces scheduling delays.

—>Parameter for Comparison

- **Real-Time Performance**
  - PID loops need deterministic timing.
  - MCUs provide predictable execution.
  - SBCs running Linux introduce scheduling delays.
- **I/O and Peripherals**
  - Need PWM, timers, encoder counters, ADC, and interrupts.
  - MCUs provide rich hardware peripherals.
  - SBC GPIO timing is less reliable.
- **Processing Power**
  - Inner PID needs fast but simple computation.
  - Outer planning may need higher compute.
  - MCUs handle control; SBCs handle perception.
- **Power Consumption**
  - Battery-powered robots require low power.
  - MCUs consume very little power.
  - SBCs consume more due to OS and CPU overhead.
- **Software Environment**
  - MCUs run bare-metal firmware.
  - SBCs run Linux and support ROS.
- **Cost and Ecosystem**
  - MCUs are cheaper and widely supported.
  - SBCs are costlier but powerful.

Part B: It is included as another file in repository