<u>UML Package Diagram and Class Diagram for Squash-Ball ROS2 Simulation</u>

We organize the system into logical packages to reflect its structure and dependencies. For example, one package might be gui (containing the GUINode), another pressure (containing PressureRecommenderNode and PhysicsCalculator), simulator (SimulatorInterfaceNode), logger (DataLoggerNode and LogEntry), and a common package for shared data types like LaunchParams. UML package diagrams group related classes and show dependencies between these packages. This high-level view simplifies a complex system by encapsulating functionality; e.g., the gui package depends on pressure and simulator (via ROS2 Services and Actions) but has no direct dependency on logger. Each package is a namespace of classes. Dependencies (shown as dashed arrows) indicate that one package (e.g. gui) uses services or topics provided by another (e.g. pressure, simulator). In practice, the common package might provide shared configuration classes (like LaunchParams) used by multiple nodes.

UML Class Diagram

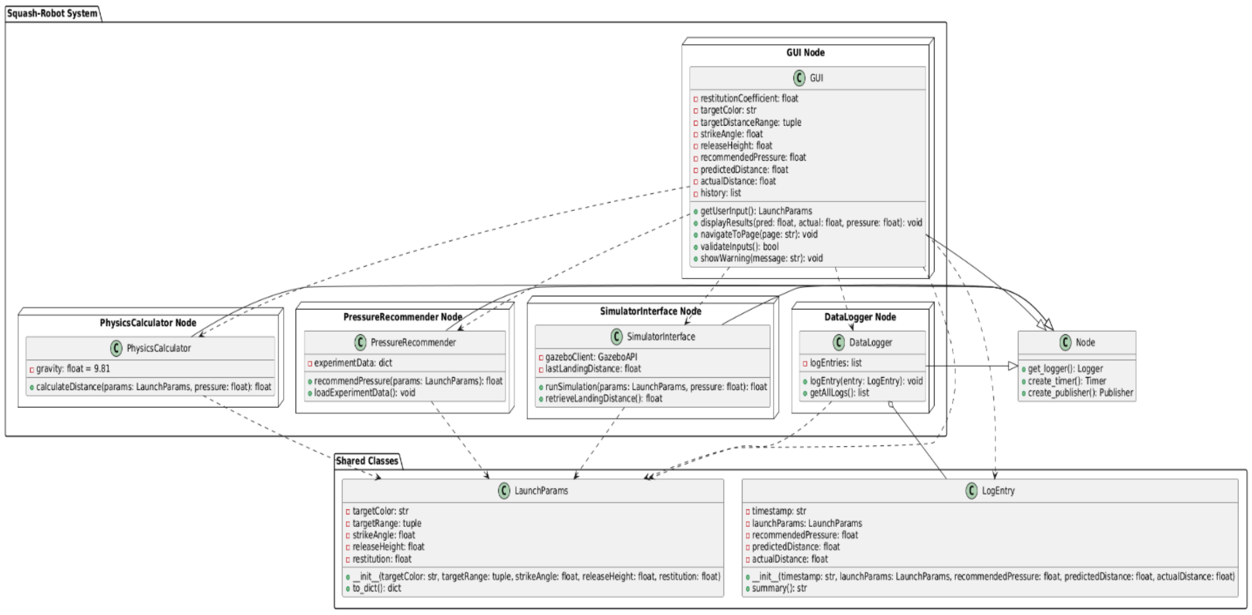
Squash-Robot System
GUI Node
GUI
restitutionCoefficient: float
targetColor: str
targetDistanceRange: tuple
strikeAngle: float
releaseHeight: float
recommendedPressure: float
predictedDistance: float
actualDistance: float
history: list
getUserInput(): LaunchParams
displayResults(pred: float, actual: float, pressure: float): void
navigateToPage(page: str): void
validateInputs(): bool
showWarning(message: str): void
PhysicsCalculator Node
PhysicsCalculator
gravity: float = 9.81
calculateDistance(params: LaunchParams, pressure: float): float
PressureRecommender Node
PressureRecommender
experimentData: dict
recommendPressure(params: LaunchParams): float
loadExperimentData(): void
SimulatorInterface Node
SimulatorInterface
gazeboClient: GazeboAPI
lastLandingDistance: float
runSimulation(params: LaunchParams, pressure: float): float
retrieveLandingDistance(): float
DataLogger Node
DataLogger
logEntries: list
logEntry(entry: LogEntry): void
getAllLogs(): list
Node
get_logger(): Logger
create_timer(): Timer
create_publisher(): Publisher
Shared Classes
LaunchParams
targetColor: str
targetRange: tuple
strikeAngle: float
releaseHeight: float
restitution: float
__init__(targetColor: str, targetRange: tuple, strikeAngle: float, releaseHeight: float, restitution: float)
to_dict(): dict
LogEntry
timestamp: str
launchParams: LaunchParams
recommendedPressure: float
predictedDistance: float
actualDistance: float
__init__(timestamp: str, launchParams: LaunchParams, recommendedPressure: float, predictedDistance: float, actualDistance: float)
summary(): str

Figure: UML Class Diagram for the squash-ball ROS2 simulation system.

In this class diagram, we clearly mark ROS2 node classes (bold names) versus plain helper classes, and we label their key attributes/methods and communications. The main node classes and helpers are:

- GUINode (ROS2 node): *Attributes:* restitutionCoefficient: float, targetColor: str, targetDistanceRange: tuple, strikeAngle: float, releaseHeight: float, recommendedPressure: float, predictedDistance: float, actualDistance: float, history: list. *Methods:* getUserInput() -> LaunchParams, displayResults(pred: float, actual: float, pressure: float) -> void, navigateToPage(page: str) -> void, validateInputs() -> bool, showWarning(message: str) -> void. This node receives user input and drives the simulation.
  It calls PressureRecommenderNode via a ROS2 Service (synchronous request) to compute optimal pressure, and sends a ROS2 Action goal to SimulatorInterfaceNode to run the long simulation task. It also subscribes to topics for status updates (e.g. landing distance) to update the UI.
- PressureRecommenderNode (ROS2 node): *Attributes:* experimentData: dict. *Methods:* loadExperimentData() -> void, recommendPressure(params: LaunchParams) -> float. This node calculates the optimal pressure (e.g. using regression or physics). It uses the helper class PhysicsCalculator (composition) to perform the physical computation of distance. It publishes the recommended pressure on a ROS2 topic (e.g. "recommendedPressure") so other nodes (GUI, logger) can receive it. Topics in ROS2 are designed for ongoing data exchange, making them ideal for streaming status updates like pressure or distance.
- SimulatorInterfaceNode (ROS2 node): *Attributes:* gazeboClient: GazeboAPI (interface to Gazebo), lastLandingDistance: float. *Methods:* runSimulation(params: LaunchParams, pressure: float) -> float, retrieveLandingDistance() -> float. This node interacts with Gazebo to simulate the ball strike at a given pressure. It provides an Action server for the long-running simulation (runSimulation) which takes the input parameters and pressure. Upon completion, it publishes the result (landing distance) on a topic (e.g. "landingDistance"), so subscribers can record or display it. Using an Action for this task follows ROS2 guidelines for long-running behaviors with feedback.
- DataLoggerNode (ROS2 node): *Attributes:* logEntries: list of LogEntry. *Methods:* logEntry(entry: LogEntry) -> void, getAllLogs() -> list. This node listens to the published topics ("recommendedPressure", "landingDistance", etc.) and creates a LogEntry record for each simulation attempt. It then stores these

entries (e.g. in memory or a file). Since ROS2 topics act as a bus for data exchange, the logger can subscribe to multiple topics without tight coupling.

- PhysicsCalculator (helper class): *Attributes:* gravity: float = 9.81. *Methods:* calculateTrajectory(params: LaunchParams, pressure: float) -> float. This class encodes the physics of the squash ball's flight. It is *not* a ROS node but is used by PressureRecommenderNode to compute distances from given pressures (e.g. solving ballistics).
- LaunchParams (data class): *Attributes:* targetColor: str, targetRange: tuple, strikeAngle: float, releaseHeight: float, restitution: float, *etc.*. This class holds all input settings (user-selected target, physical constants, initial conditions). It can be constructed from GUI input and passed through services/actions to other nodes.
- LogEntry (data class): *Attributes:* timestamp: str, launchParams: LaunchParams, recommendedPressure: float, predictedDistance: float, actualDistance: float. *Methods:* formatRecord() -> str.
  Each LogEntry encapsulates a full result record of one simulation trial.
  The DataLoggerNode creates and stores one of these entries per run, capturing both predicted and actual outcomes.

The UML class diagram shows directed associations for composition (solid diamond) and usage (dashed arrows). For example, PressureRecommenderNode has a composition relationship to PhysicsCalculator (it contains or uses an instance of PhysicsCalculator). DataLoggerNode similarly composes LogEntry instances. The dashed arrows between nodes are labeled with the ROS2 communication type: a dashed arrow from GUINode to PressureRecommenderNode labeled "Service: requestPressure()" denotes a service call. A dashed arrow from GUINode to SimulatorInterfaceNode labeled "Action: runSimulation()" denotes sending an action goal. Dotted arrows from PressureRecommenderNode and SimulatorInterfaceNode back to GUINode (labeled "Topic: recommendedPressure" and "Topic: landingDistance") represent topic publications that GUI and logger subscribe to. As ROS2 docs note, services suit quick request/response use-cases, while actions handle longer tasks with feedback, and topics are used for ongoing data streams.