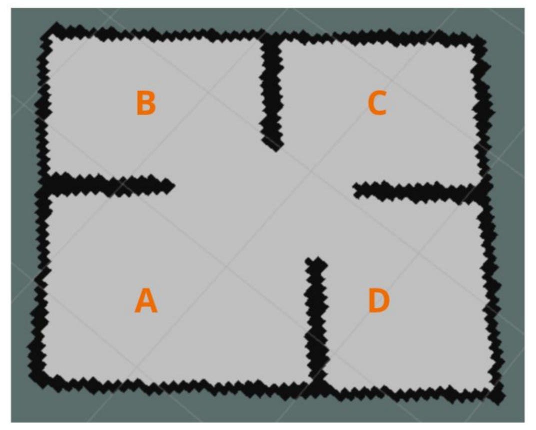
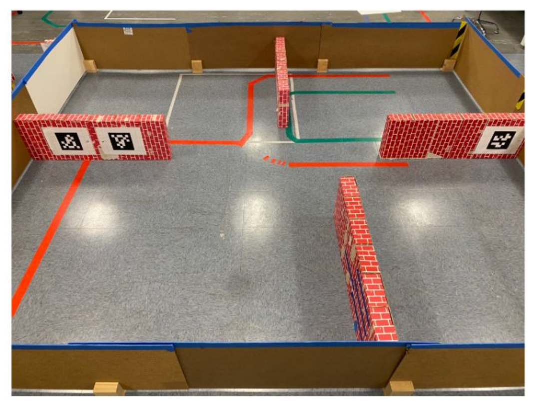
Multi-Robot Home Surveillance

James Lee
5(F) May 2023

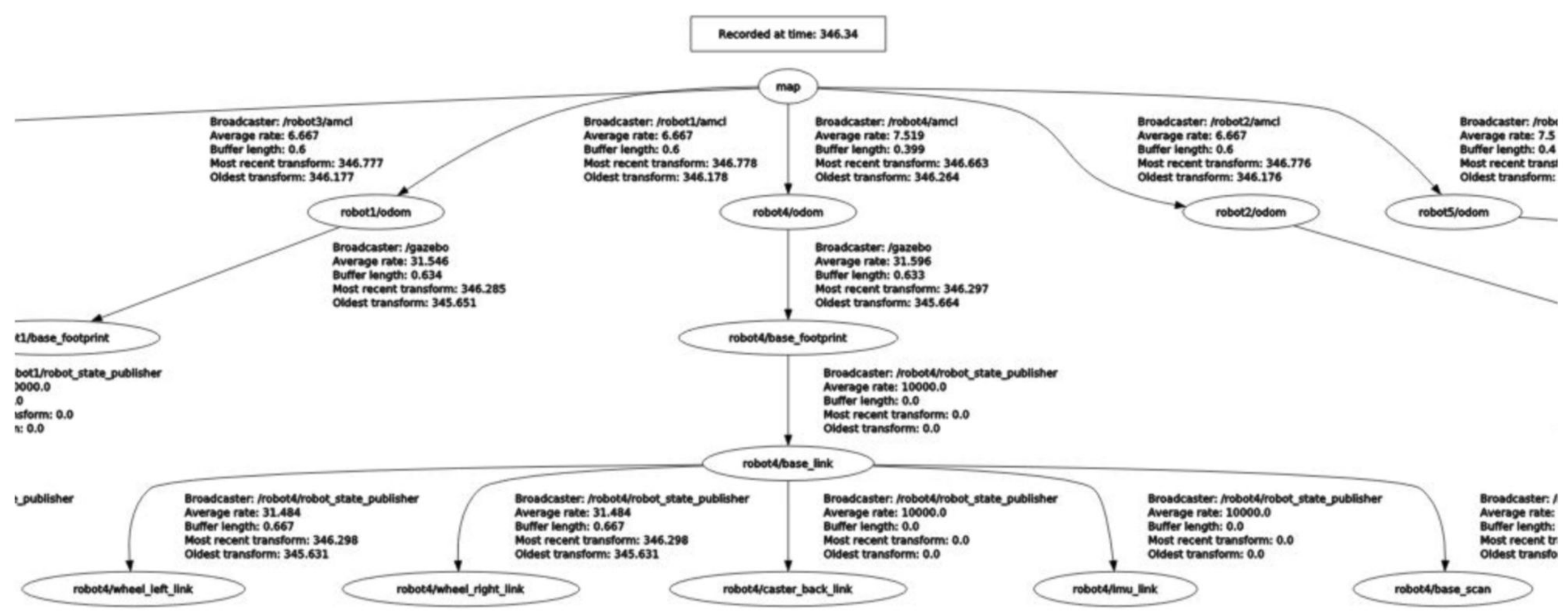
Problem Statement

- The home to the right is divided into four security zones.
- We can think of them as a sequence, ordered by their importance: [A, B, C, D].
- If we have *n* live robots, we want the first *n* zones in the list to be patrolled by at least one robot.
  - n== 2 → patrolled(A, B);
  - n == 3 → patrolled(A, B, C)
  - 1 <= *n* <= 4
- The goal is for this condition to be an invariant for our program, despite robot failures.

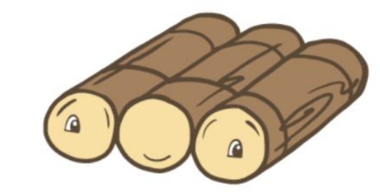




Navigation - The TF Tree



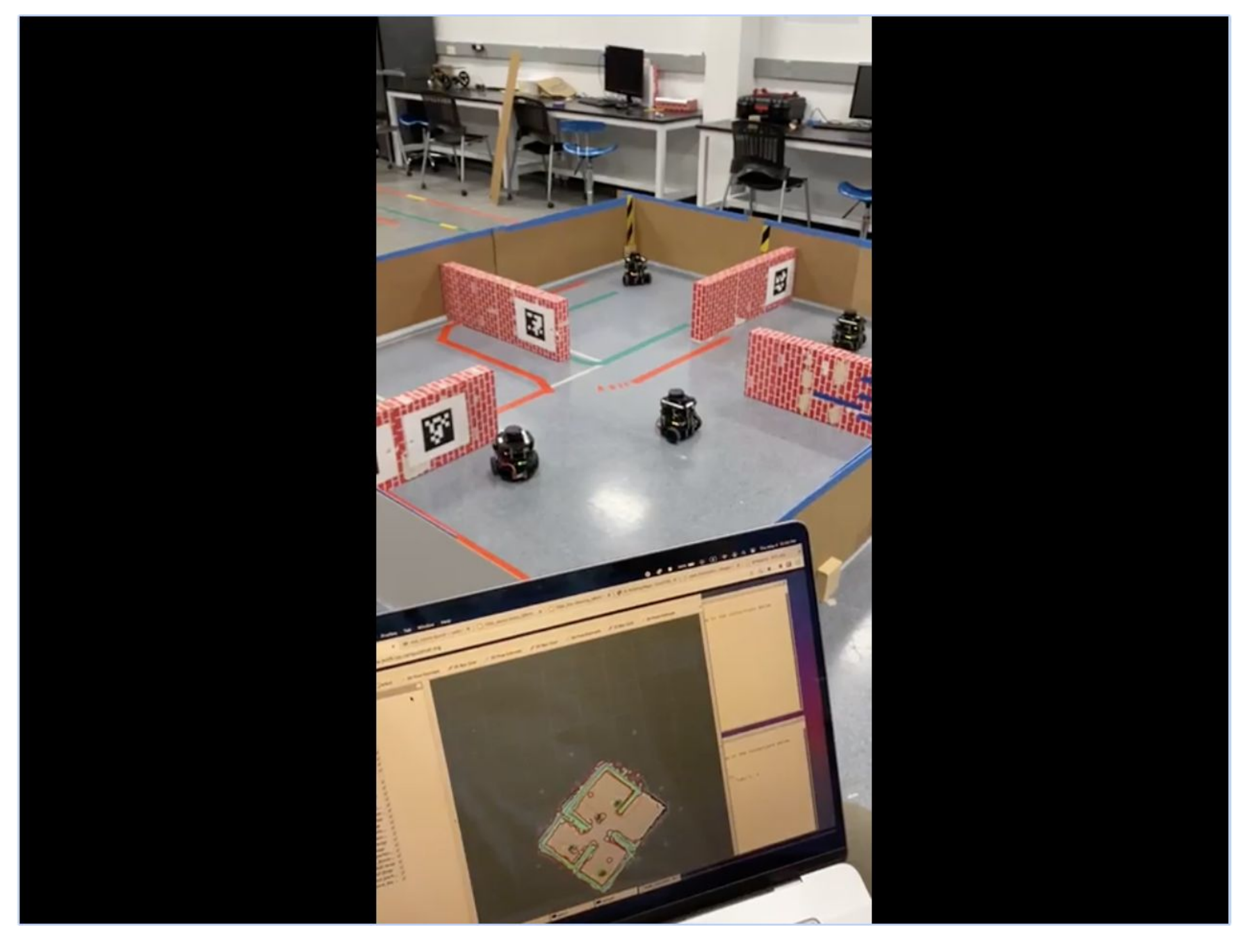
LEA: Implementation I



1. Background:
   a. Each of the robots is represented as an mp (member of parliament) node.
      i. *mp_roba*, *mp_rafael*, *etc.*
   b. Each node is in one of three states: *follower*, *candidate*, or *leader*.
   c. If a node is a leader, it starts a *homage_request_thread* that periodically publishes messages to the other nodes to maintain its status and prevent new elections.
   d. Every mp node has a term number, and the term numbers of mp nodes are exchanged every time they communicate via ROS topics.
   e. If an mp node's term number is less than a received term number, it updates its term number to the received term number.

Challenges - I

- We need to:
  - map the home
    - SLAM (Simultaneous Localization and Mapping) with the *gmapping* package
  - have multiple robots patrol the security zone in parallel
    - AMCL (Adaptive Monte Carlo Localization) and the *move_base* package
    - One thread per robot with a SimpleActionClient sending *move_base* goals
  - have the system be resilient to robot failures, and adjust for the invariant appropriately
    - Rely on Raft's Leader Election Algorithm ('LEA') to elect a *leader* robot.
    - The remaining robots will be *followers*.
    - The leader is responsible for maintaining the invariant among followers.



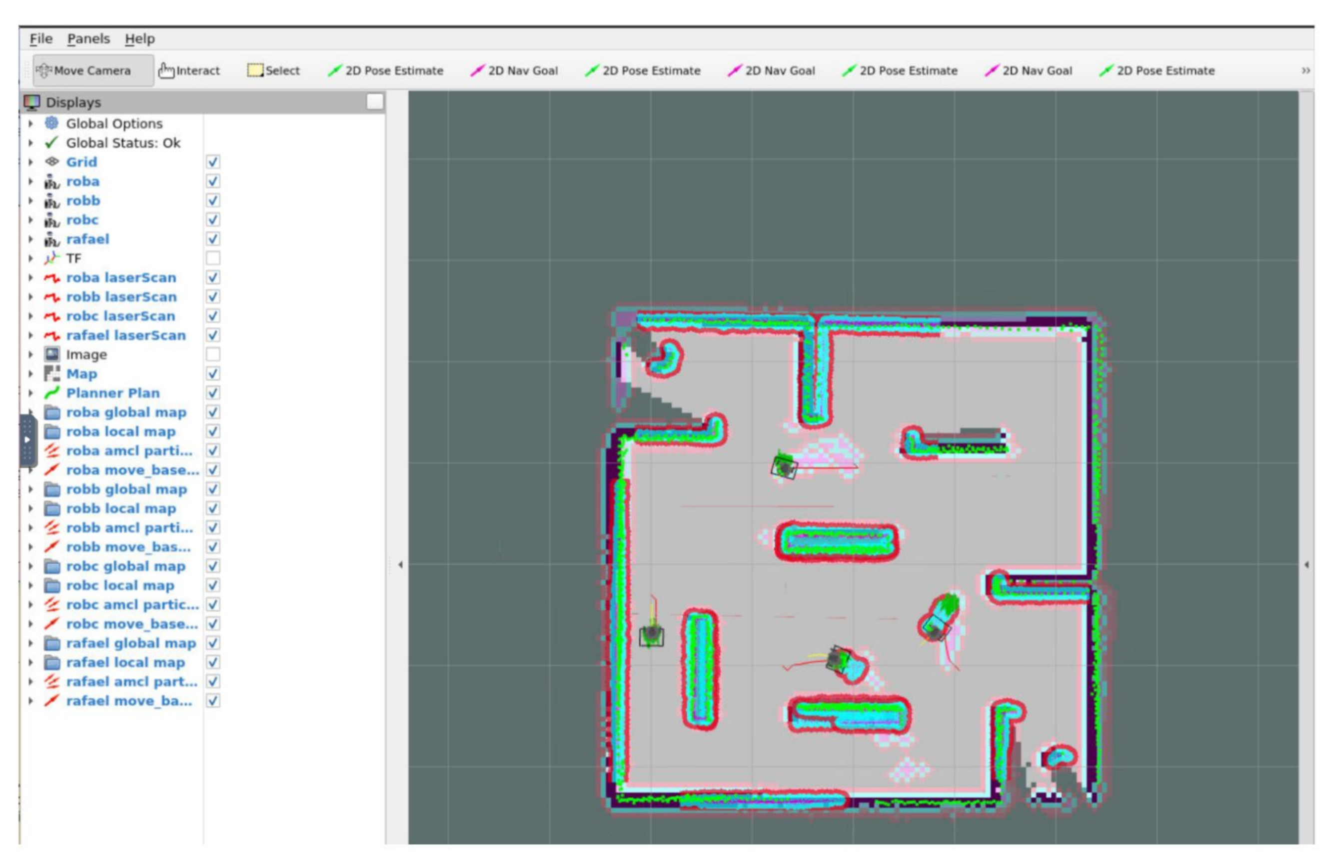
Limitations - I

- The mp nodes are all running on the remote VNC computer, which also runs the *roscore*.
  - So killing the mp node is just a simulation of robot failure, not true robot failure.
    - We're just killing a process on the vnc computer, not the robot itself
  - We can remedy this by binding the mp node's life to any other nodes that are crucial to our robot fulfilling its task.
    - E.g., for our project the */roba/amcl* and */roba/move_base* nodes are critical.
    - So we can consult *roscore* periodically to see if any of these nodes die at any given point. And if they do, we can kill *mp_roba*.
    - We can also define and run more fine-grained custom nodes that keep track of any functional status of the robot we're interested in, and bind the life of those custom nodes to the mp nodes.
- Running each mp node on its corresponding robot is a bad idea!
  - Only if a robot gets totally destroyed, or the mp process fails, will the algorithm work.
  - Performance will take a hit:
    - bandwidth must be consumed for messages between mp nodes
    - robot's computational load will increase

Challenges - II

- More on system resiliency:
  - The leader always patrols zone A, which has the highest priority.
  - On election, the leader assigns the remaining zones to followers to maintain the invariant.
  - On leader failure:
    - global interrupts issued to all patrol threads (all robots stop).
    - the LEA elects a new leader
  - On follower failure:
    - local interrupts issued by leader to relevant patrol threads (some robots stop).
    - Example:
      - n==4 → patrol(A, B, C, D)
      - $robot_B$ dies
      - leader interrupts $robot_D$
      - leader assigns $robot_D$ to zone B
      - n==3 → patrol(A, B, C)

Navigation - *RViz* Interface



Limitations - II

- Our system has a single point of failure; the VNC computer running roscore.
  - So we have to keep the VNC safe from whatever hostile environment the robots are exposed to.
  - Maybe this can be remedied in ROS2, which apparently does not rely on a single *roscore*.
- Our patrol algorithm is very brittle and non-adversarial
  - Depends on AMCL and move_base, which do not tolerate even slight changes to the environment, and which do not deal well with moving obstacles.
  - There are no consequences to disturbing the patrol of the robots, or the robots themselves (e.g. no alarm or 'arrests')