A Reactive Infrastructure for Swarm Robotics

By Emma Pollak (RBE & CS),
Filippo Marcantoni (RBE)
Lorenzo Manfredi Segato (RBE & CS)
Advisor: Professor Carlo Pinciroli (RBE & CS)

Why PiELo?

PiELo explores designing a language from the ground up for swarms. To show the benefits, we will compare to ROS, a standard robotics programming framework.

Limitations of ROS:
- Difficult to scale to multiple robots
- Reactivity and consensus require setting up multiple nodes and callback functions

This code example shows a synchronization behavior across a swarm. PiELo makes this a significantly easier task in terms of code quantity and complexity.

```cpp
bool barrierCallback(barrier_test::Barrier::Request &req,
                     barrier_test::Barrier::Response &res)
{
  std::unique_lock<std::mutex> lock(barrier_mutex);

  // Record arrival (ignore duplicates)
  if(std::find(arrived_ids.begin(), arrived_ids.end(), req.robot_id) == arrived_ids.end()){
      ROS_INFO("Robot %d arrived at the barrier.", req.robot_id);
      arrived_ids.push_back(req.robot_id);
  }
  else {
      ROS_WARN("Robot %d has already signaled the barrier.", req.robot_id);
  }

  // Wait until all arrived
  while (arrived_ids.size() < robot_count) {
      cv.wait(lock);
  }

  // When all arrived, release each waiting call
  res.success = true;
  res.message = "Barrier released for robot " + std::to_string(req.robot_id);
  ROS_INFO("Barrier released for robot %d", req.robot_id);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "barrier_server");
  ros::NodeHandle nh;

  // Advertise barrier service
  ros::ServiceServer service = nh.advertiseService("barrier", barrierCallback);
  ROS_INFO("Barrier server is ready. Waiting for all robots to arrive.");
  ros::AsyncSpinner spinner(5);
  spinner.start();

  // Monitor barrier condition, notify waiting callbacks.
  ros::Rate rate(1);

  while (ros::ok()) {
      {
          std::unique_lock<std::mutex> lock(barrier_mutex);
          if (arrived_ids.size() >= expected_count) {
              cv.notify_all();
          }
      }
      rate.sleep();
  }
  return 0;
}
```

ROS: 5 files with 200+ lines

```
(begin
(include functions.txt)
(var global inert barrierComplete)
(set barrierComplete 0)
(var global inert barrierSum)
(var map reactive barrier)

(fun reactive checkBarrier () (begin
    (var local inert barrierSum)
    (set barrierSum 0)
    (foreach inert (in i barrier') (set barrierSum (+ barrierSum i)))
    (if (> barrierSum 1) (set barrierComplete 1) 0)))

(var local inert total_distance)
(set total_distance 10)
(set distance_covered (get_distance_covered))
(set barrier (if ( ≥ distance_covered' total_distance) 1 0))
(set barrierChecker (checkBarrier))
(set speed (if (&& ( ≥ distance_covered' total_distance) (! barrierComplete)) 0 (* (+ (% robotID 5) 1) 2)))
(set leftWheelVelocity speed')
(set rightWheelVelocity speed')
(spin)
(fun reactive step () (set distance_covered (get_distance_covered))))
```

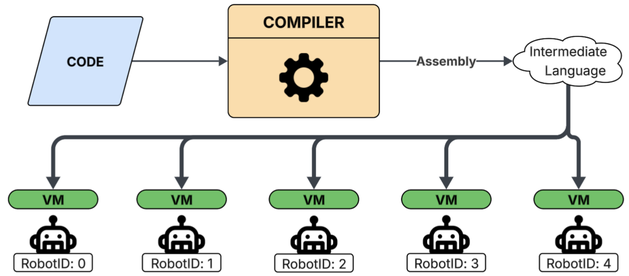
PiELo: 22 lines of code

The PiELo Programming Language

PiELo is a programming language for swarm robotics.

Objectives:
- Trivial scaling for many robots
- Easily respond to events such as sensor updates or swarm communications
- Automatically achieve consensus across the swarm

Code Compilation



Copies of the code are distributed to each robot in the swarm

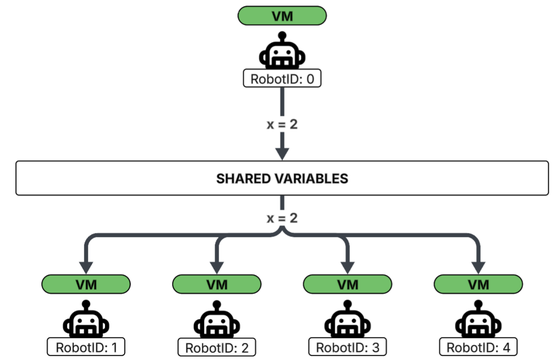
Practical Implementations

Barrier Test

Driving Robot

Distributed Architecture

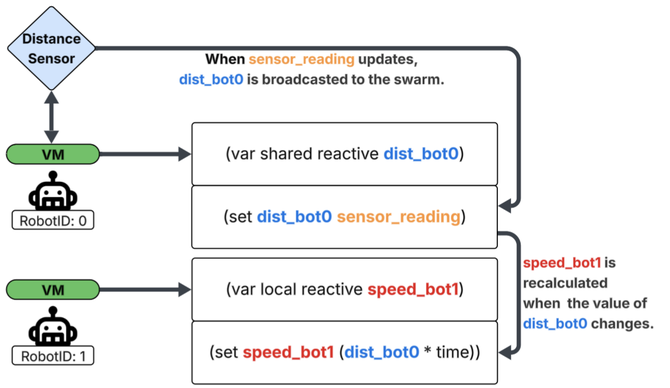
PiELo enables inter-robot communication using a network of Virtual Machines executing code on each robot. Shared variables are automatically updated with the rest of the swarm.



Distributed Virtual Machine

The swarm achieves consensus with only two lines of code.

With reactivity built in, responding to updates across the swarm is trivial.



Reactivity & Consensus