Threshold-based Task Allocation

Florian Zimmermann (IN)
Daniel Camilo Rojas Quirós (EDIC)
Bojan Karlaš (IN)
What’s the project about?

• A swarm of 5 E-Pucks must handle events

• The events are red cylinders and are detected with the E-Puck camera

• Target:
  • Minimize energy/#handled_events

• We tested two task allocation schemes:
  • Fixed Threshold
  • Variable Threshold

• The aim is to study the problem in Webots and in real E-Pucks
How did we do it?

• Simple image processing for identifying events

• The stimulus is evaluated periodically

• 3 state FSM

• Variable Threshold:
  • $T_{\text{active}}$ increases if an event is handled, otherwise is reset
  • Keeps active the robots when they are able to handle events
Event detection

Goal: Assess local stimulus

Initial design:

• Take a picture
• Count the number of red pixels
• Rotate by the same angle as camera FOV
• Repeat for one full rotation
• Local stimulus = total number of red pixels over all images
Event detection

**Problem:** camera isn't centered on the axis of rotation of the robot

→ Images overlap in the distance, events may be counted multiple times

Final design:

• Continuously take pictures while rotating
• Calculate stimulus once per picture taken
• Become active as soon as stimulus is above threshold
Event following

• Continuously take pictures

• Count red pixels for each column of the image

• Previous column doesn't contain red pixel, current column does → start of event

• Previous column did contain red pixel, current column doesn't → end of event

• Total number of red pixels between start and end of event → size of event

• Adjust direction towards biggest (=closest) event

• No red pixel in image → do random walk
Event handling

More than half of image pixels are red → robot is in front of event
• No obstacle avoidance
• Only straight movement

Less than half of image pixels are red → event has been handled

Each robot keeps count of #events handled
How we detect red pixels

• Absolute threshold (if R>X) works, but is sensitive to lighting/shadows
How threshold impacts performance

- Constant number of events
- Fixed threshold
- Lower settings are better because we always have new events coming
Other parameters that we’ve studied

How does Active Duration impact performance?

• Performance metric only concerned about efficiency

• Lower active duration:
  • Robots go to idle often and reassess the scene – more careful

• Increasing gets more events handled, but up to a limit
Other parameters that we’ve studied

How does Scan Period impact performance?

- Scan period doesn’t have much impact on performance

- Still, having scans more often gets more events handled
Comparison between **fixed** and **variable** Active Duration

• With Variable Active duration:
  • Slight performance increase
  • Robots adapt
  • They travel less, but turn out to be more efficient
Distribution of work among robots

- Robots not equally active, but difference is not large
- We can also see that robots that handle less events, travel less
Hardware implementation

• Didn’t go as smooth as the Webots version
• Main issue:
  – The E-Pucks reset continuously when moving (every 3s)
  – Our theory:
    • Bluetooth + camera + motors = High current consumption => voltage drop => uC reset
  – The only solution we found: stop the E-Puck to use the camera
  – This handicapped our algorithms
Other issues found

• The delays were generated with counters in the main loop
  – ~ Non-uniform duration

• The E-Puck wheels are red
  – Another source of noise

• The use of real E-Pucks involve delays not represented in Webots
  – Start and stop the wheels
  – Take a picture (few ms to ~1.5s)

• The camera is not ideal
  – It’s more sensitive to green than to red and blue
Thank you!

Questions?