Multiple Drone/Robot Navigation and Synchronization Platform

Mentor: Professor Sangjin Hong

Kimberly Gilot, Raymond Li, Adrian Ortega, and Shaan Panjwani

Stony Brook University
Electrical and Computer Engineering

INSYNQ

Introduction

Robots have become quite an integral part of modern day society as they aid people in performing daily tasks, thus allowing for a more efficient experience. InSynq's goal is to create a platform allowing for the synchronization of multiple virtual robots in order to allow more effective navigation to supplement and aid human activities. This project's objectives are to have simulated robots be able to navigate in both an environment that contains static and dynamic obstacles, with each robot being controlled concurrently by a centralized navigation server.

Approach

Our project was broken down into 3 parts:

1. Simulated Robots/Obstacles – Implemented using the **SystemC** Library (C++).

2. Graphical User Interface (GUI) – Implemented using **Qt** GUI Framework to create a desktop application for Windows.

3. Server-Client Model – Implemented using Qt Server and **WinSock** Client.

Simulation Specifications

We simulated the environment using the following parameters:

- 4 robots
- 2 dynamic obstacles
- 9 rows and 10 columns
- Maximum speed of 2m/s for robots
- Obstacle speed of 4m/s
- Simulated clock (Δt) is 1ms
- Robot proximity sensor sensitivity set to 3m
- Maximum 1 robot per grid space
- 60 grid spaces

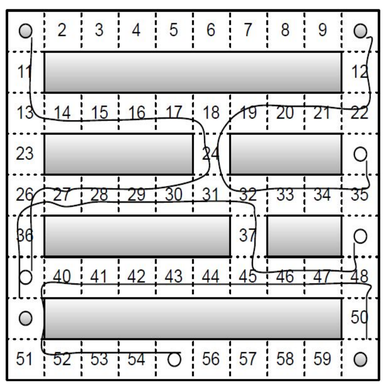

Figure 1: A 60-grid map with robot paths drawn.

Concept and Design

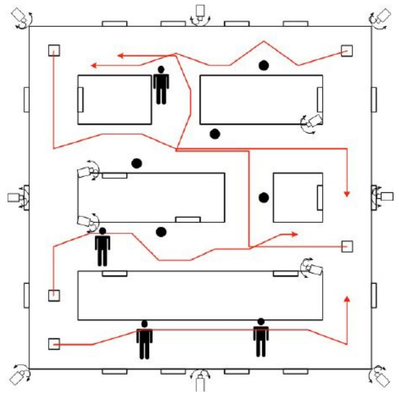

Figure 2: A sample environment for InSynq.

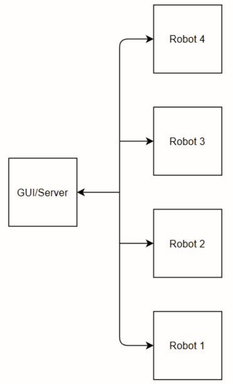

Figure 3: The Robot/Server communication model.

	Robot ID	urrent Gri	Next Grid	Status
1	1	10	12	MOVING
2	0	1	11	STOP
3	2	49	39	MOVING
4	3	60	50	MOVING

Figure 4: Robot-status table.

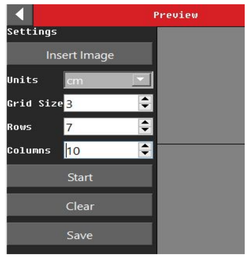

Figure 5: Grid size and configuration settings.

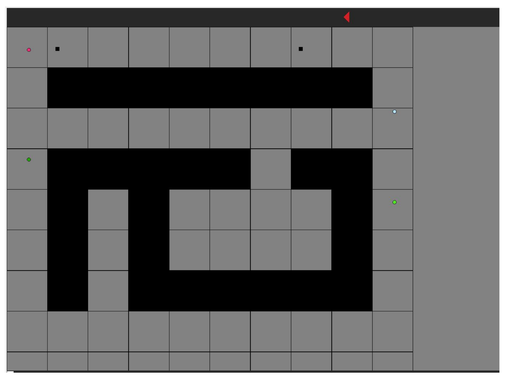

Figure 6: This is a functioning simulation that includes robots (circles) and objects (squares).

The user interface features:

- Save/load map layouts..
- Ability to select numbers of rows, columns, and their sizes.
- Shows status table with general robot information.
- Ability to select and deselect grid lines.
- Display a main grid with moving robots and obstacles.

Implementation

Server-Client Communication:

- Robots and obstacles are treated as clients and **multithreaded** using SystemC threads.
- Robots send data to the server in the following cases:
  * Boundary approached.
  * Obstacle detected/cleared.

Speed Control:

- To prevent **deadlocks**, robot speed can be determined beforehand so that they arrive at **critical nodes** at separate times.
- A node-based data structure is used to hold which robots go to which critical nodes and when, and the speed is adjusted accordingly.

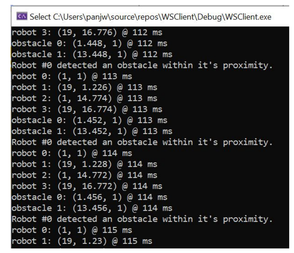

Figure 7: Console window showing robot and obstacle positions and status.

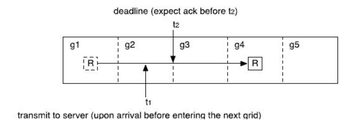

Figure 8: Robot-Server boundary communication.

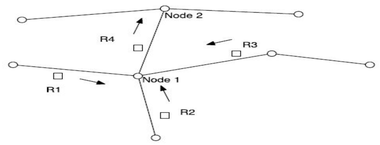

Figure 9: Node structure.

Glossary

Critical Nodes: Nodes that are intersections between two or more hallways where the possibility of deadlock is greater.
Deadlock: A case where two or more robots want to go into each other's grids, and as a result, all are stuck.
Multithreading: The ability to have multiple threads of execution concurrently.
Qt: Cross-platform application development framework.
SystemC: An event-driven simulation interface enabling concurrent processes using C++ syntax.
WinSock: Windows Socket API for network services.

Acknowledgements

- Professor Sangjin Hong
- Professor Harbans Dhadwal

InSynq