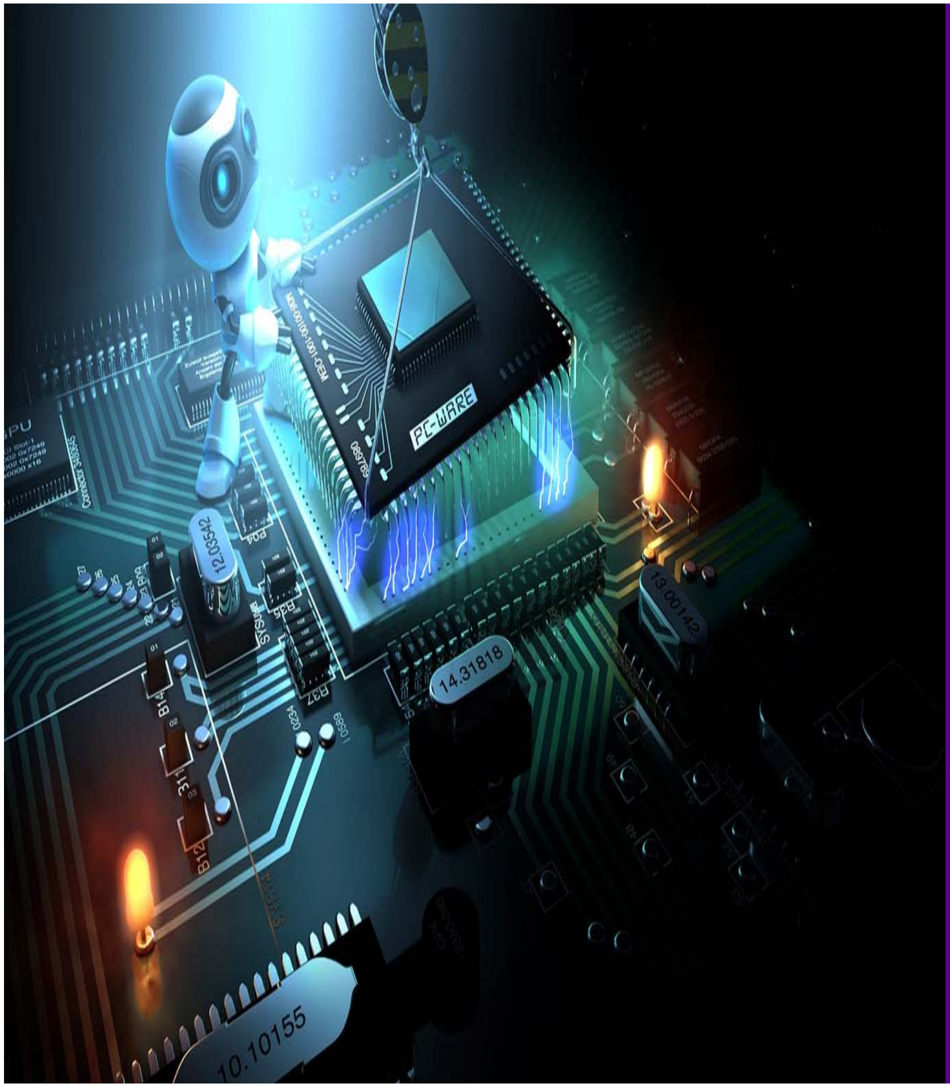
NYU

ME-GY 6933

Advanced Mechatronics

Mohit Lala
Shweta Vaviya



NYU

TERM PROJECT

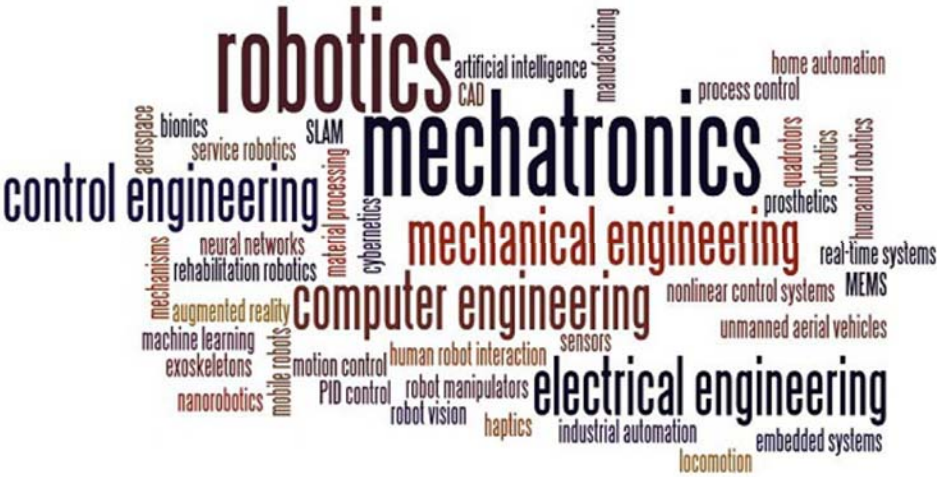
"Swarm robots for environment mapping"

A swarm of two mobile robots which localizes itself in an unknown environment and generates an approximate map using various sensors and algorithms.

NYU

Applications

- Mapping of unknown environment and locating land mines during military operations.
- Surveillance in dangerous and inaccessible places.
- Rescue search operations during disaster management.
- Sensor specific application such as using the system to sense the levels of toxic gases, etc.
- For assistance in mining operations.



NYU

Our Approach

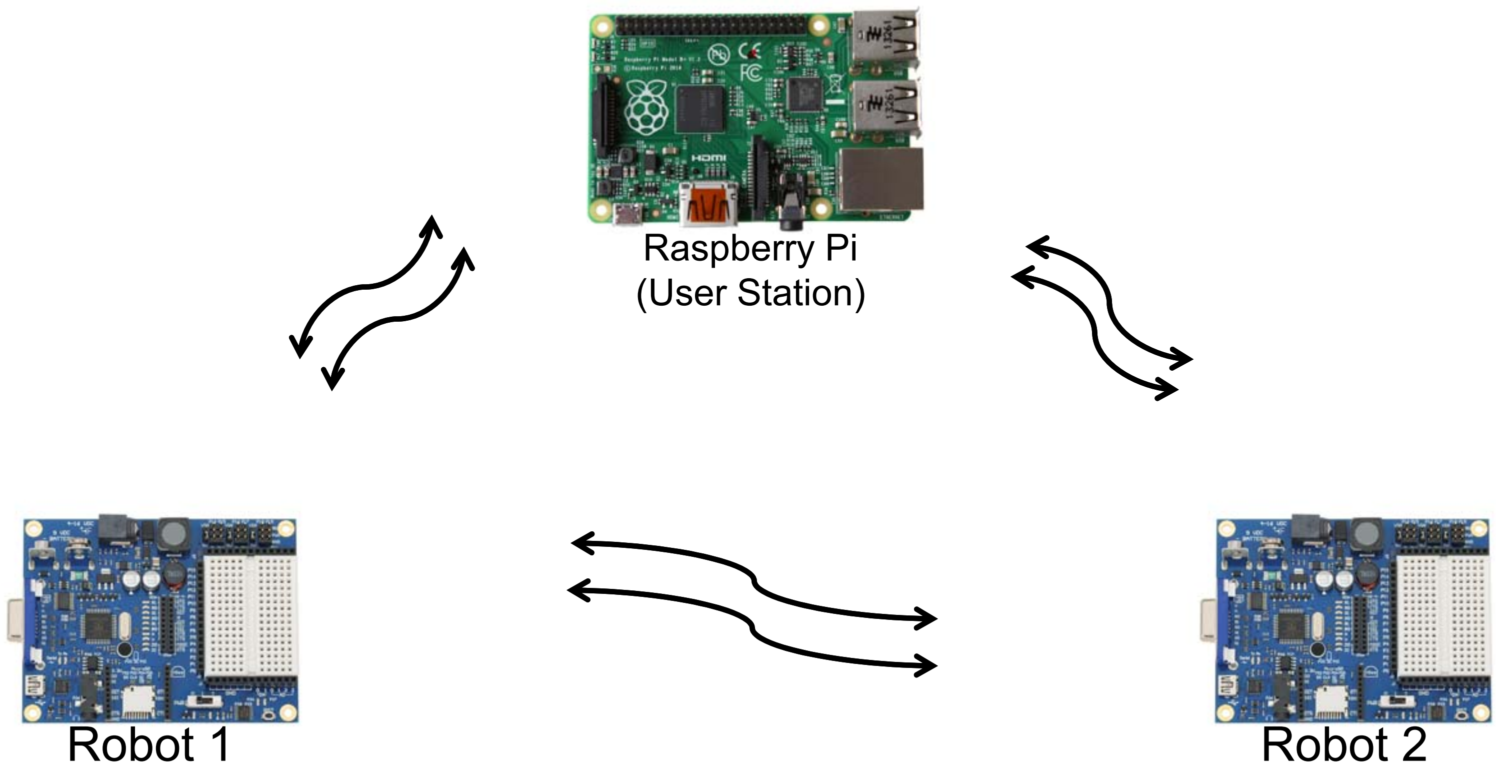
- Our approach towards this project was structured based on the curriculum structure for Advanced Mechatronics course
- The first prototype generated 2D maps of an environment using a wall-follower algorithm implemented on Arduino Uno integrated with range sensors
- The second prototype was then modeled on a Parallax Propeller board where a swarm of two robots was employed to navigate an unknown environment using Wavefront and BFS algorithms
- The final prototype generates a map of the environment on a Rapsberry Pi 3 board which effectively communicates with the robots to obtain map and robot localization data from the propeller board mounted on each robot

NYU

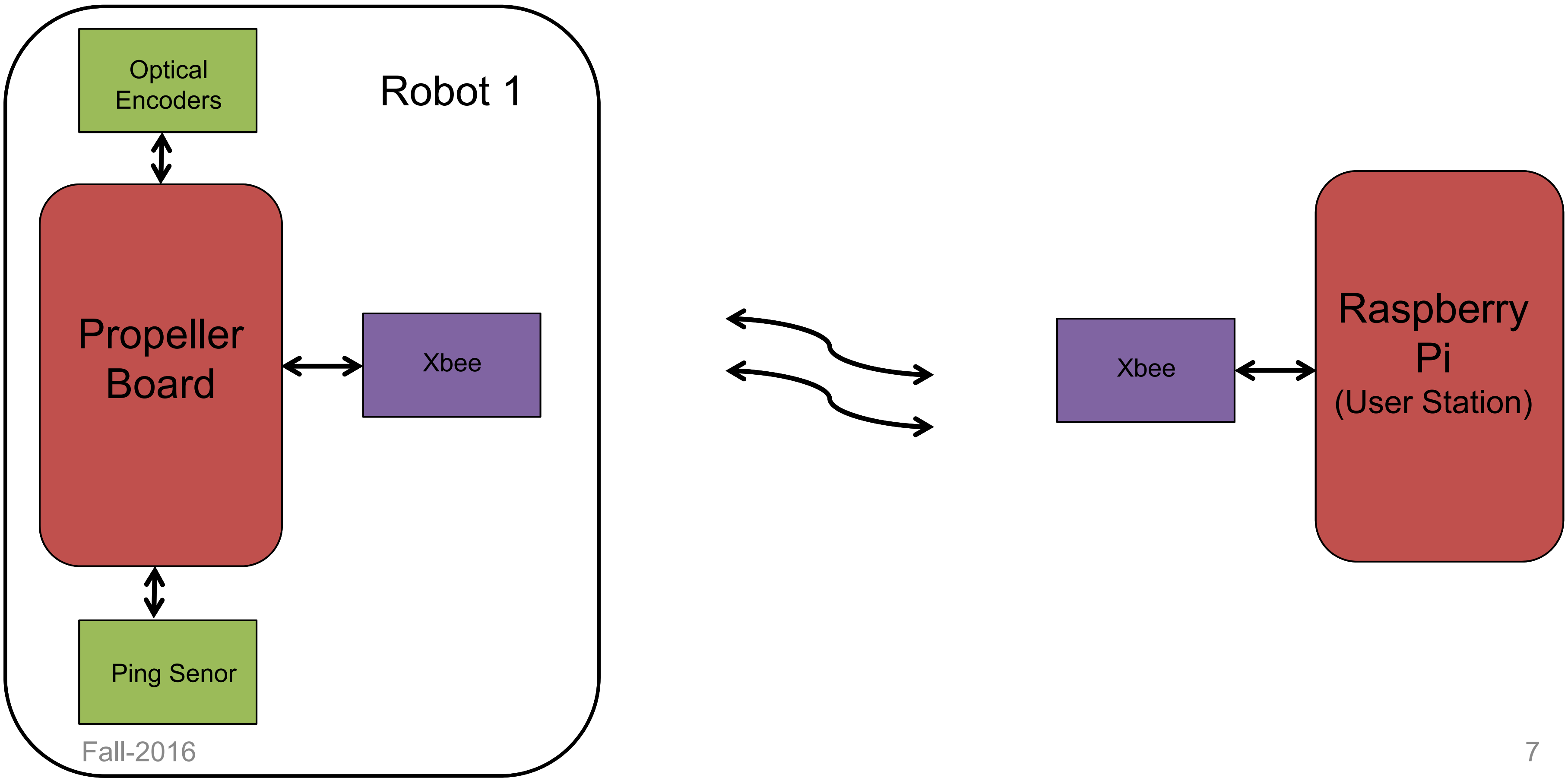
Hardware

- Development boards:
  - ✓2 x Propeller board of education
  - ✓1 x Raspberry Pi 3
- 2 x Ping Sensors
- 4 x Optical encoders
- 3 x Xbee Series 1 modules
- 4 x 6V DC Motors
- 2 x L293D Motor drivers
- 4 x 5V batteries

System Block diagram

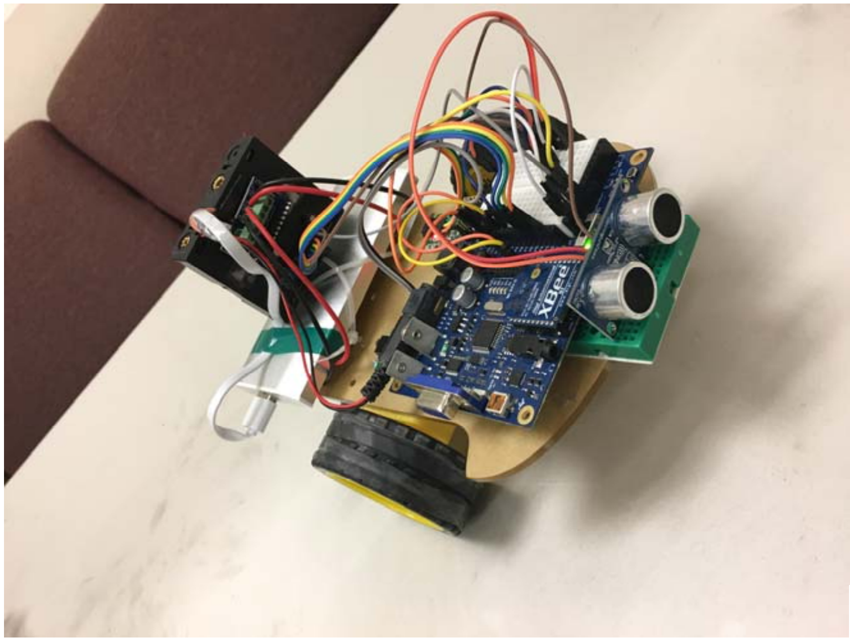


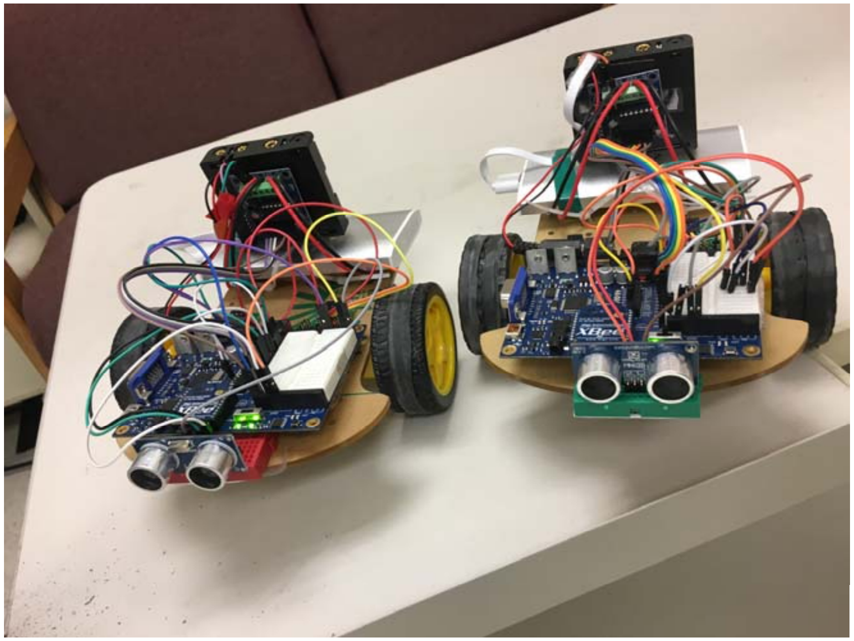
Detailed Block Diagram



NYU

Robots



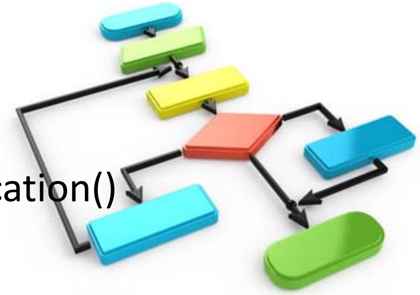


NYU

System Features

- Algorithm and the code developed are generic enough to be used with different systems with similar development board functionalities
- The code is modular and robust enough to be easily manipulated for hardware connections and additions
- The number of robots can be increased with minimalistic variable initialization changes
- System generates simulation of the robot motion on the serial terminal which is helpful in debugging the system
- Can broadcast localization information with respect to the map, to any computer or other robot in its network with proper configuration
- Once the map is complete the robots can be queried to autonomously reach a specific location or user defined location

On-board Algorithm for each Robot

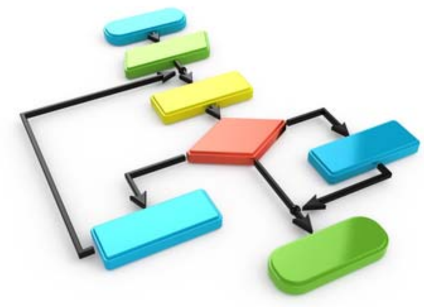


- Receive input for the function. //Generate Map() or Go to a particular Location()

Go to a location finding the shortest path()

- Receive Input for Goal Location
- Generate grid of known environment
- Assign value to each cell of the grid map using Wavefront algorithm
- Find shortest path using Breadth-first search (BFS)
- Manipulate robot motion as per the deduced path obtained by BFS
- Using appropriate feedback sensors, track the location of the robot

On-board Algorithm for each Robot

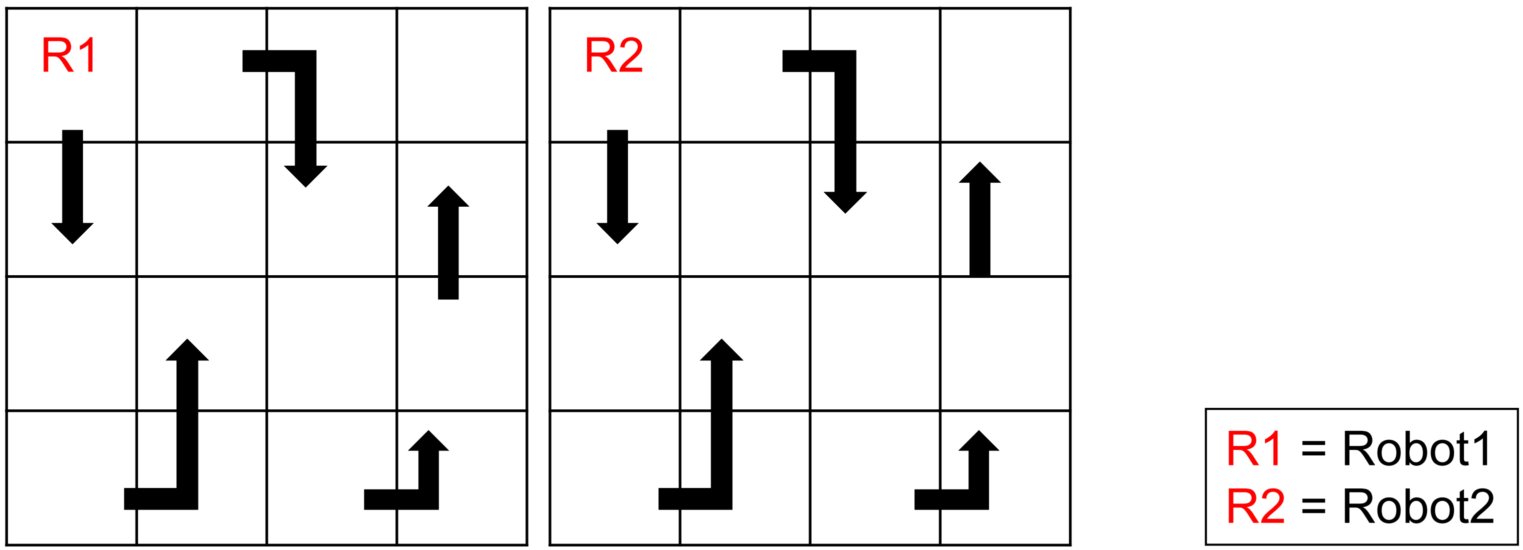


Generate Map()

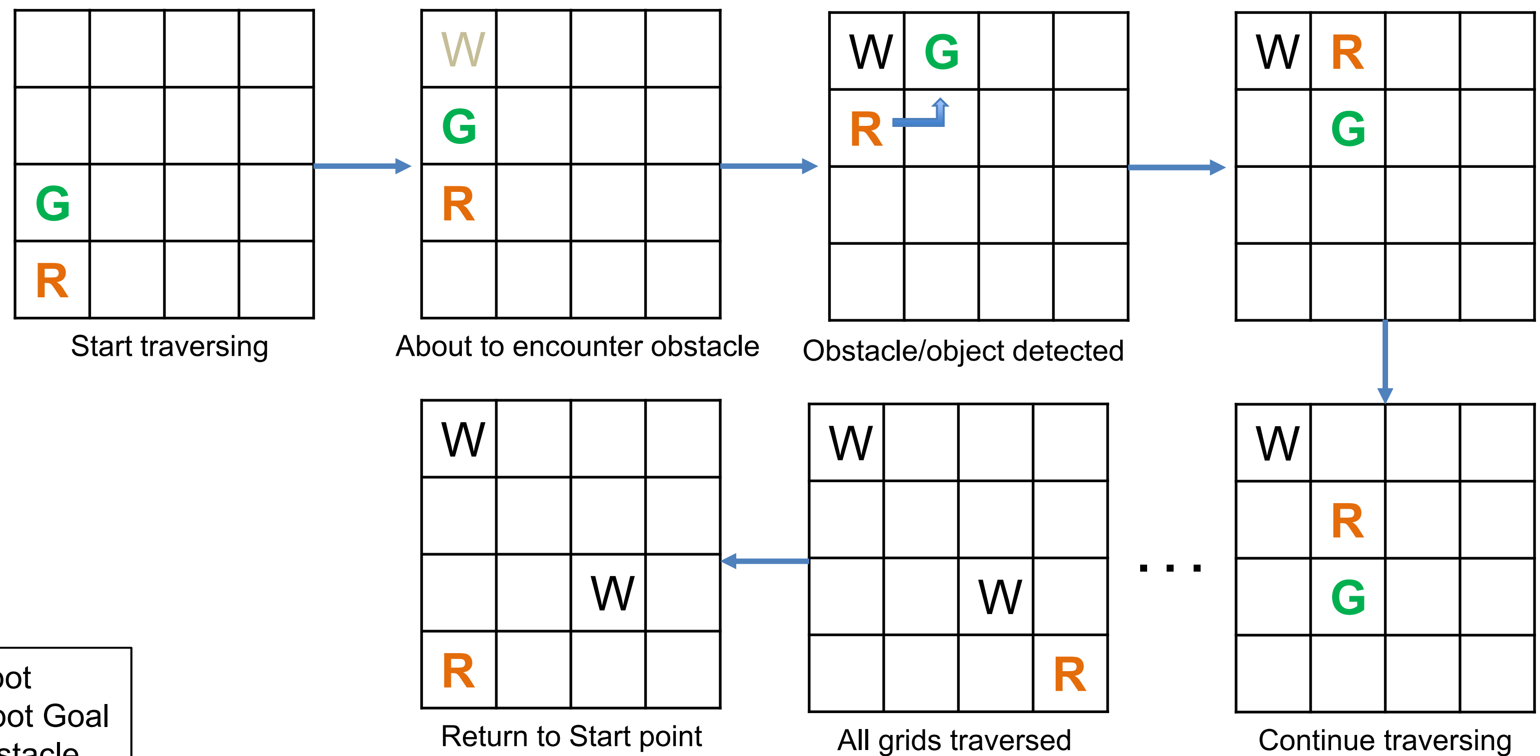
- Divide the given environment limits in grids
- Navigate to each grid and sweep the whole environment
- Send Robot location to Raspberry Pi via Xbee
- If obstacle/objects are detected, update the map
- send object location over to Raspberry Pi

Traversing the Map [Logic]

- The Robot divides the given area to be mapped in grids
- Below the map sweeping logic for a 4 x 8 grid



Generating Map



R -> Robot
G -> Robot Goal
W -> Obstacle

Find Shortest Path

Breadth-first search (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors.
We utilize the BFS algorithm to fill the grid of the map like a **Wavefront** until the robot is found on the map and this helps the Robot to determine best path to get to the Goal location.

		G			
W	W	W			
			W	W	W
			R		

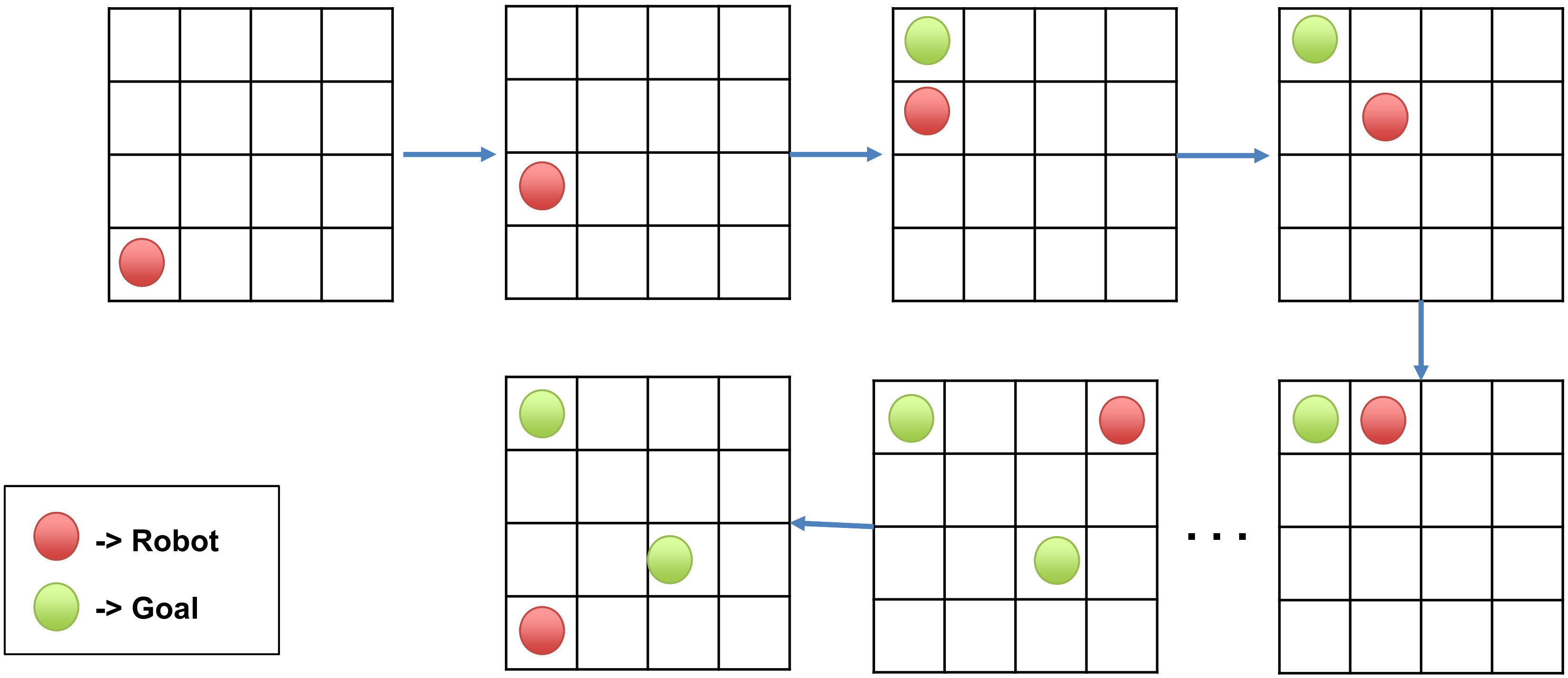
		G	2	3	4
W	W	W	3	4	5
7	6	5	4	5	6
8	7	6	W	W	W
	8	7	8		
		8	R		

		G	2	3	4
W	W	W	3	4	5
7	6	5	4	5	6
8	7	6	W	W	W
	8	7	8		
		8	R		

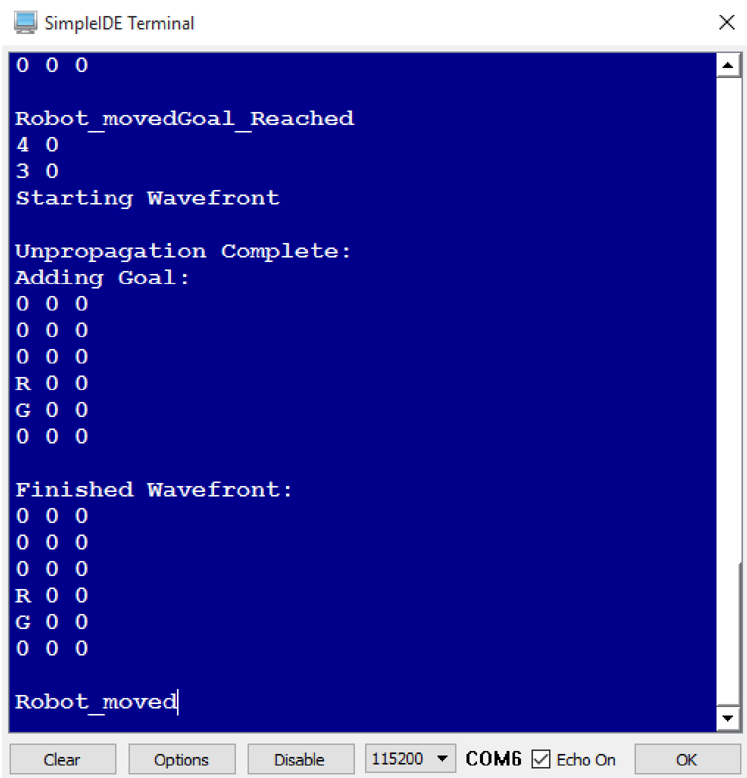
R -> Robot
G -> Robot Goal
W -> Obstacle

Localization of Robot

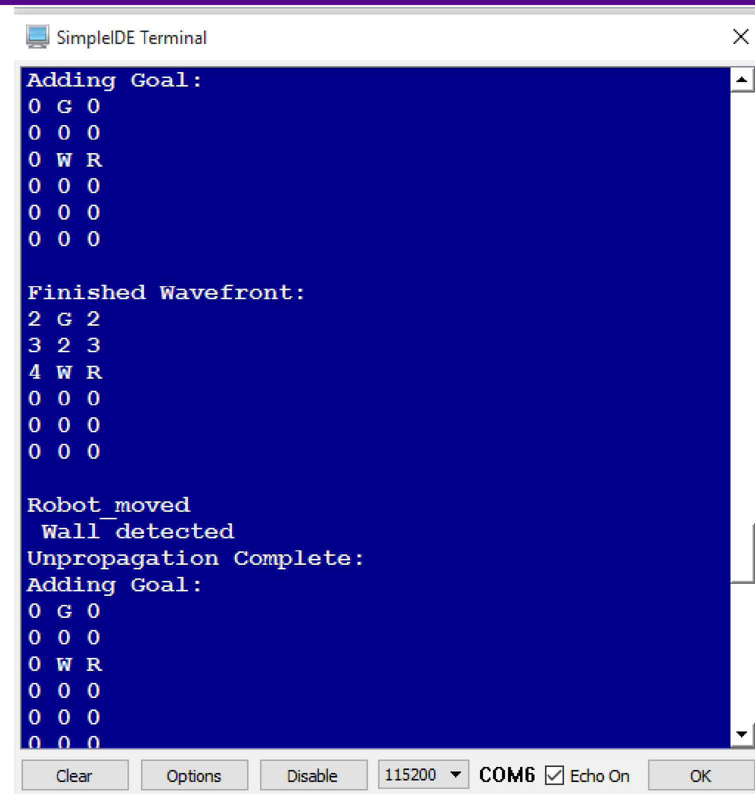
Localization of the robot:



Simulation OUTPUT on Serial Terminal



Robot (R) traversing the environment Towards Goal (G)

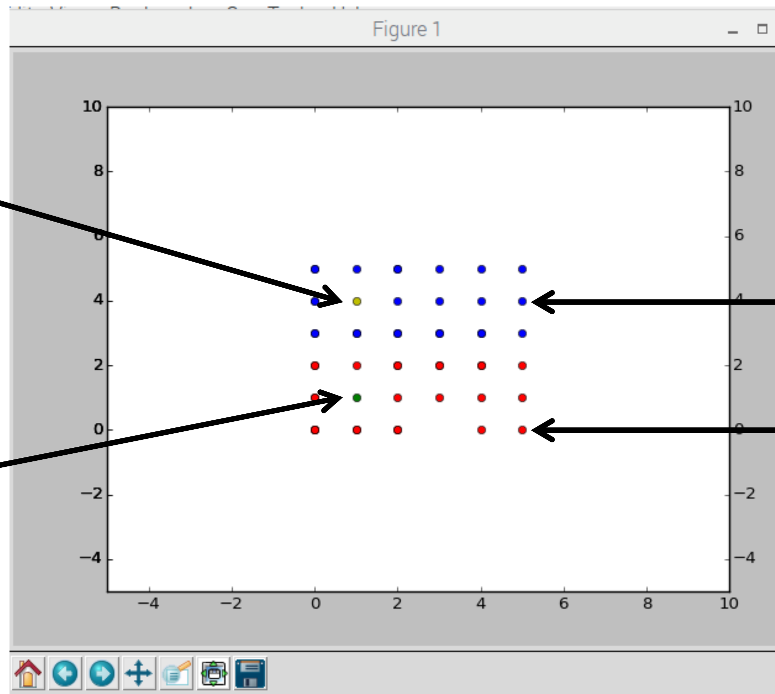


Wall (W) detected and Robot (R) travelling

Map OUTPUT on R-PI

Yellow plots are generated by Robot 2 for Obstacles/Objects

Green plots are generated by Robot 1 for Obstacles/Objects



Blue plots are generated by Robot 2

Red plots are generated by Robot 1

Grid Map generated on Raspberry Pi

Improvements over Previous Prototypes

- Introduced -> Raspberry Pi board to the system
- Improvised -> an update algorithm for Mapping
- Improved -> performance for navigation of robots
- Better communication between the robots and the user station

What We Learnt

- Hands-on experience with Propeller Board of Education and Raspberry Pi 3 board and their on-board functionalities
- Revision of C, C++ programming concepts with use of Simple IDE
- Research of available recursive algorithms and data structures for robot navigation
- Studying the simpletools.h, fdserial.h , serial.h, simpletext.h, ping.h, SharpIR.h libraries
- Configuring Xbee modules to work in a multipoint network
- API and AT modes and use of Xbee module as a co-ordinator, end device and router
- Experience with and programming of wheel encoders, selecting motor drivers and motors and research of IMU sensors to get feedback from a mobile robot
- Hands-on experience of working with Python, MATLAB, OpenCV, VNCServer

Future Scope and Limitations

- Improve feedback
  -> Employing IMU sensor and Rotary encoders instead of optical encoders to keep a track of odometry of the Robot

- Better mapping techniques
  -> Employing algorithms and controllers to generate more accurate maps in both 2D and 3D

- Improved Control system
  -> Employing the use of filters and controllers in our system model to establish error correction introduced due to environment

NYU

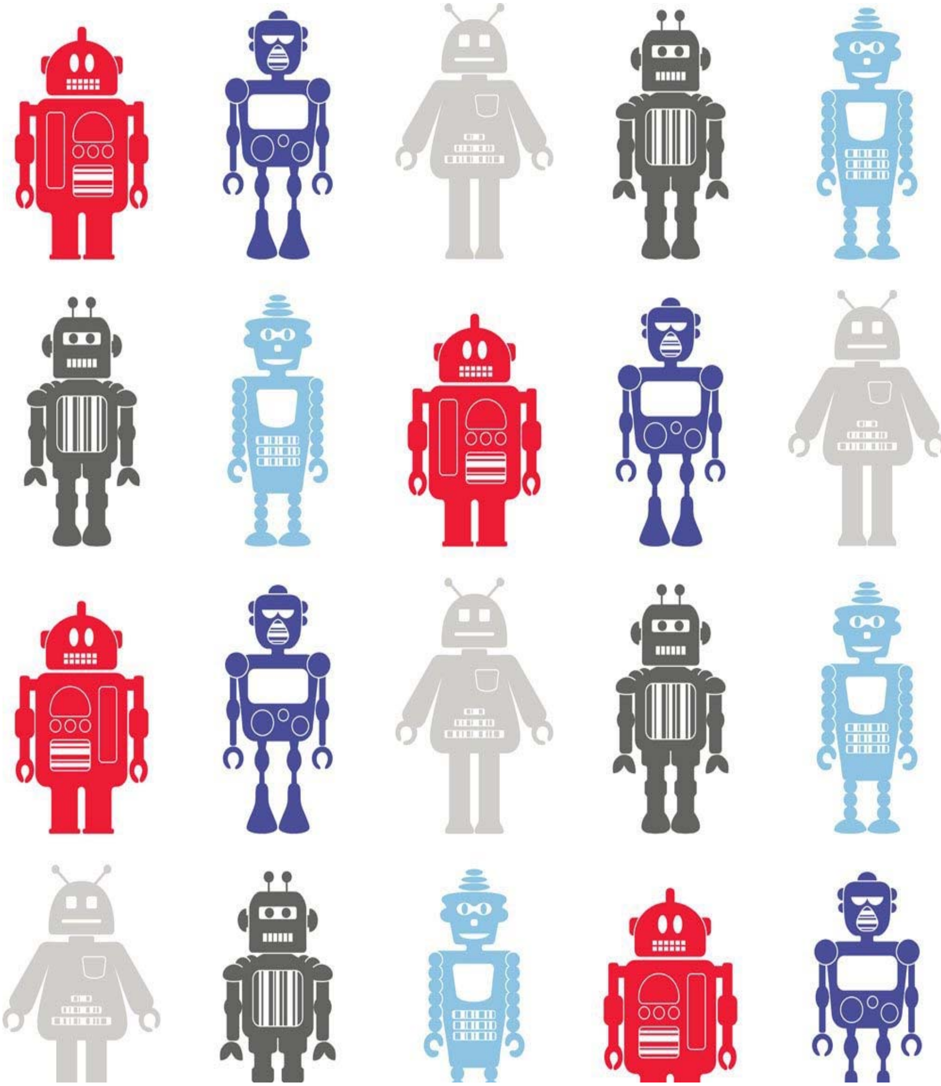
Tips & Tricks Employed

- Multi-core functionality of the propeller boards can be used as a substitute for interrupts
- Local variables can be shared over multiple tabs by defining them as ‘extern’
- To get readings from optical encoder while turning, make the wheel move with a PWM value high enough so it doesn’t slip and low enough so the optical encoder doesn’t skip a count
- Choose/Fix a castor in such a way that it doesn’t end up leading or influencing the robot direction
- For multipoint communication over Xbee if you do not want to configure the Xbee modules in your code, using AT commands, every time you want to send data to a different address, configure the Xbee with DH = 0x00 and DL = 0xFFF which makes it send data to all modules in the same network
- Motors receiving power from the development boards may cause the board to reset every time a lot of current is drawn and hence a different power supply or a supply strong enough for both the development board and motors should be used

NYU

References

- http://www.societyofrobots.com/
- http://learn.parallax.com/tutorials/
- https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf
- https://www.stackoverflow.com
- CLRS – Introduction to Algorithms
- Class notes on Propeller Intro Lec5 To 8
- Class notes on Raspberry Pi-Intro-Lec 11 to 12



NYU

THANK YOU