UNIVERSITY OF PENNSYLVANIA

ESE 350 Project Report

Multi-Robot Cooperation for Area Covering

Yizheng He, Caroline White

5/9/2013

Table of Contents

I. Introduction

Multi-robot area covering has emerged as a field of interest due to its widespread applications, including agricultural uses, large store vacuum cleaning, and search and rescue operations. Especially in situations in which the area to be covered is known, such as department stores that need to be vacuumed, localization is one method with which multiple robots could work together to cover an area. In this scenario, one of the primary barriers to efficiency is the robots' battery levels. In other words, how can you get these self-localizing robots to work together to cover an area, taking into consideration the possibility that one or some of them will run out of battery halfway through the job?

In this project, we have built a multi-robot relay system of self-localizing robots that can work together to cover a given area, even as they run out of "battery" halfway through their tasks. One of the major features of the robots was their localization capabilities. Using a network of infrared LED's on the ceiling, the robots were able to use mounted infrared sensors to determine their coordinates in the given area. A relay system was used to switch between "tiles" of infrared lights as the robot moved from quadrant to quadrant in the four-tile system. In addition, each moving robot was able to communicate with the "master" microcontroller using mounted ZigBee modules.

Our final implementation of the project consisted of two robots working together to zigzag over an area. Each robot had an artificial battery level that would go down as it moved across the area, and as soon as the first one ran out of battery, it would move to the designated "charging station," and the next robot would move to the coordinate at which it left off and finish the zigzagging.

II. Hardware

Hardware List:

- Pololu 3pi
- mbed NXP LPC1768 microcontrollers
- Microsoft Wii Remote infrared cameras
- MRF24J40 ZigBee modules
- Infrared LEDs
- 1kΩ resistors
- Mini breadboards
- AAA batteries
- 120 V power supply

Master Device:

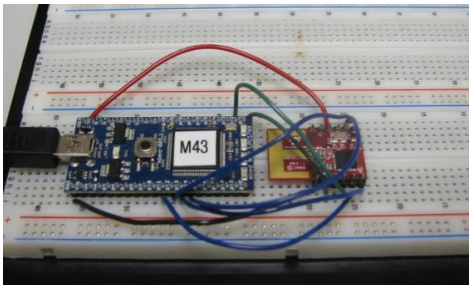

Figure 1: The Master Board

The "master" consisted of an mbed microcontroller mounted on a breadboard and connected to a ZigBee module and a power supply. The ZigBee module allowed it to communicate with the moving robots and to send commands and receive data. Because the master module was stationary, we were able to keep it connected to a PC and output all of the data values that were being sent and received by it – a great tool for both debugging and demonstrating the artificial "battery level" of the moving bots.

Slave Devices:

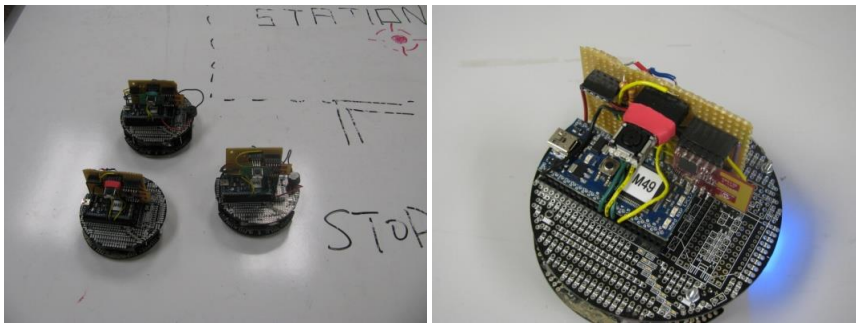

Figure 2: The Slave Robot(s)

The “slave” robots were Pololu 3pi robots with additional microcontrollers, sensors, and modules mounted on them for localization and communication purposes. Each Pololu 3pi robot had a shield with an mbed, Wii Remote infrared camera, and ZigBee Wi-Fi module mounted on it.

All of the code for the slave robots was loaded onto their mbed microcontrollers. The mbeds served as the control center for all other aspects of the hardware – it coordinated the communication via ZigBee, received the sensor data, and controlled the motors on the Pololu.

The ZigBee modules allowed each of our “slave” robots to communicate with the “master” microcontroller. The module enables two devices to communicate with each other over a range of different frequencies. By setting each of our “slave” robots on a different channel (with a different frequency), we were able to allow the master microcontroller to communicate with different robots for the same task. In the future, we could allow the slave robots to communicate with each other directly using the ZigBee modules so that they don’t have to use the master as a go-between.

The Wii Remote infrared sensors were chosen as the localization sensors due to the way in which they recognized infrared light sources. In particular, the sensors picked up on the relative locations of the four brightest infrared points in range, and in this way we were able to set up our infrared network to ease our localization calculations.

Localization System:

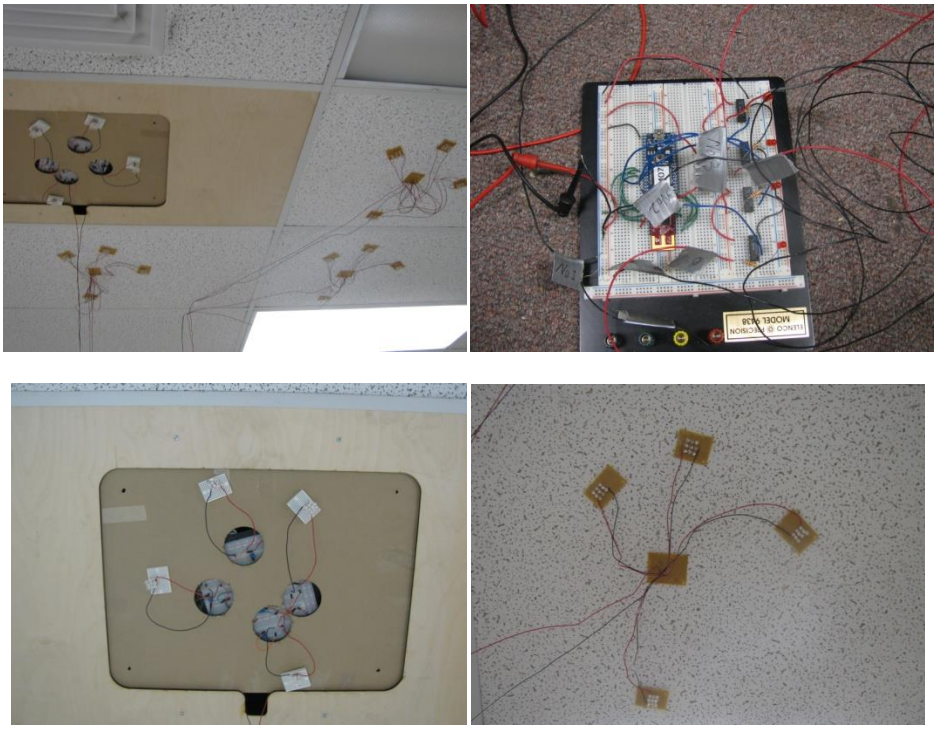

Figure 3: The Localization Tiles and the Localization Controller

Our localization system consisted of a four-tile infrared LED network on the ceiling, as well as a relay system that switched the power from one tile to another.

Each of the four LED tiles mounted on the ceiling consisted of four clusters of nine infrared diodes arranged in an asymmetric pattern. The asymmetry is what allows the slave robots to determine their position when their infrared sensors pick up on the four brightest points – based on the angle between each light point, every location on the board gives a unique reading.

III. Software

Software Overview

The code for this project is divided into three parts: slave, master and localization controller, all implemented on mbed microcontrollers. The slave code is installed on two Pololu 3pi robots. It runs the vision, localization, motion and communication modules so that the robots can receive command from the master and finish the zigzag tasks. The master code is installed on the controller mbed. It supervises the status of the slave robots and sends commands to them to implement the team cooperation. All the team mechanism is implemented in the master code. The localization controller is an independent module. It communicates with the slave robots and controls the on and off of the localization tiles for them. All the communications are through the MRF24J40 ZigBee wifi module.

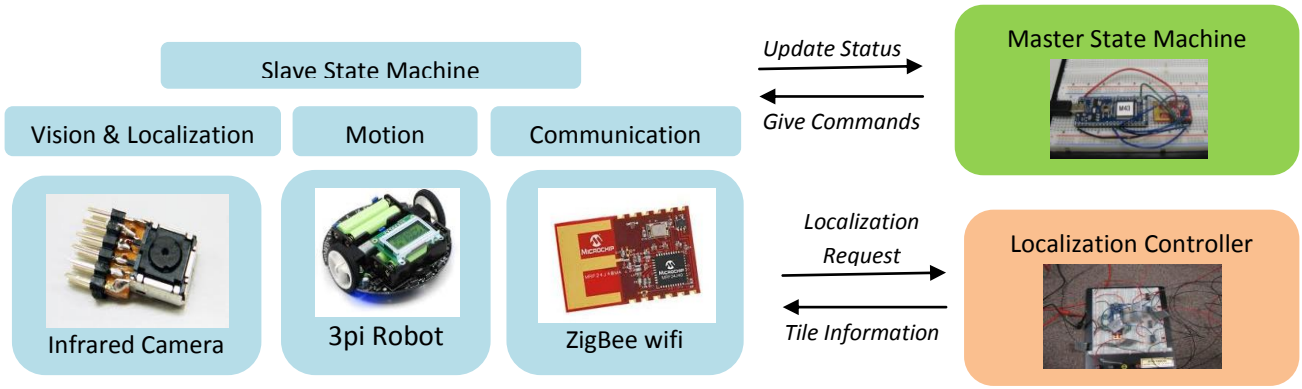

Figure 4: System Overview

Slave

Vision & Localization

The vision code (Vision.h) drives the Wii Remote infrared camera to detect infrared points from the environment. The Wiimote cameras are internally coded such that when requested, they return the positions of four strongest infrared sources in the environment. The data are in the format of x and y coordinates, and the origin is the bottom left corner of the camera screen. The Wii Remote camera connects to the mbed through i2c buses. The main function in the vision module sends the request (some fixed HEX codes) to the camera through the bus and waits for the results.

The Localization code (Localization.h) uses information from vision to calculate the coordinates (x, y, direction) of the robot in a world system. There are four localization tiles in this project, each covering an area on the field. The localization algorithm first calculates the position of the robot in the local tile system (its position in the area covered by the tile it is under). This process is enabled by the fact that the shape formed by the four IR LEDs in each tile is asymmetric. The algorithm distinguishes distinct LEDs based on the distances between every two of them. With the positions of four distinct points in the robot's local frame, the robot's position in the global frame defined by a single LED tile can be calculated. Then, depending on which tile the robot is under, a relative position vector is added to generate the

world coordinate. Since the relative positions of the localization tiles are fixed, these position vectors can be measured in a calibration process and stored as constant integers in the code.

Localization Controller

The robot knows which tile it is under from the Localization Controller (Loco-controller.cpp). When a robot moves out from a tile area, the camera can no longer find the LEDs in the previous tile and will return some random huge numbers. The localization code on the robot detects such errors and sends out an error message to the localization controller. The latter then turns off the previous localization tile and turns on another one. It keeps switching between different tiles until the robot stops sending error message, which means the robot can localize again in the new tile. The controller sends the ID of the new tile (each tile is given an ID) to the robot so that the localization code on the robot can pick the correct relative position vectors.

Motion

The Motion module (Motion.h) sends commands to the motors. It can make the robot go to a certain spot on the field or do a zigzag to cover certain areas. These commands can be directly called when the slaves receive command from master and start to work.

Communication

The Communication module (ZigBee.h) drives the MRF24J40 ZigBee wifi. It enables the robot to send and receive messages from the master and localization controller. The mbed interfaces with MRF24J40 through serial port. The low level API was developed before and open source. The communication module consists of a sending function and a receiving function, which sends and receives strings from the serial port in a fixed format. The messages are then broadcast by the ZigBee antenna. The ZigBee module also supports channel selection, which facilitates multi-robot communication. In this project, slave robots send messages to the master on different ZigBee channels. The master switches between channels to command different robots. The communication between the robots and the localization controller is on another channel, so that different types of messages do not interfere with each other.

Slave State Machine

The state machine (Slave.cpp) incorporates all the above functionalities in a closed the loop so that the robot can listen to and finish the task sent from the master. Since the state machine is closed loop, the program can make sure the robot will behave as expected.

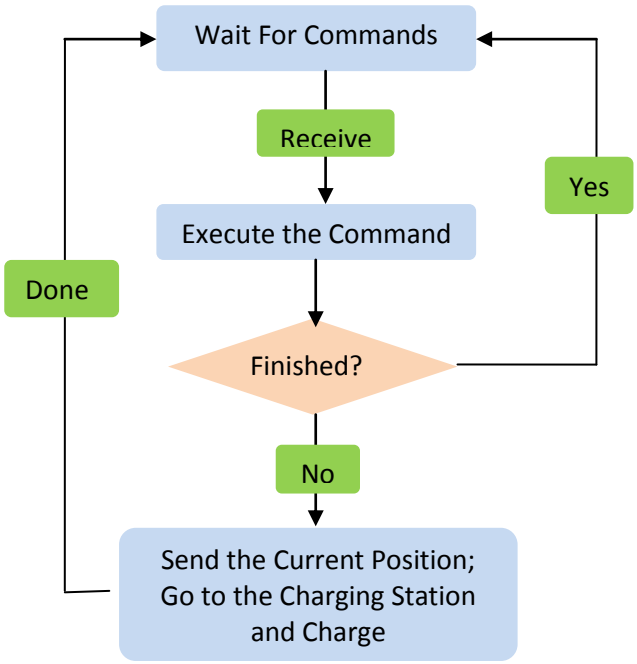

Figure 5: Slave State Machine

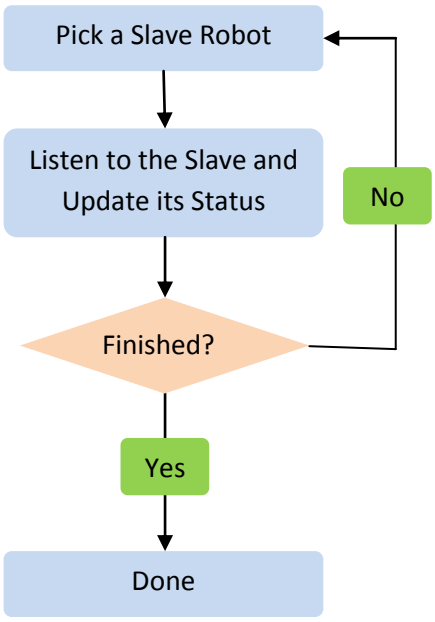

Figure 6: Master State Machine

Master

The master code includes a communication model (same as the one in the slave) and a controlling state machine (Master.cpp). The state machine implements the cooperation mechanism: it first picks the robot in best condition (with the highest battery level) to start the task. Then when the robot in work is running low of battery, the state machine will send the location where it stops to another robot. The loop continues until the task is done.

IV. Conclusion

Our project fulfilled all of the major requirements that we set out to fulfill at the beginning of the semester:

1. The robots are able to localize themselves in a known environment using a network of mounted infrared LEDs on the ceiling.
2. The robots are able to communicate with each other to divide the area to be covered among them. This was implemented using a “master” controller that sent out instructions and multiple “slave” robots that did the actual area covering.
3. We implemented an artificial “battery level” that was used to imitate one robot taking over the job after another one died out. The robot that ran out of batteries was able to move back to a designated “charging station” to recharge while the next robot continued the work.

Moving forward, there are many directions in which we, or future researchers working on a similar issue, can take our project. One aspect that we didn’t get a chance to explore was minimizing the amount of time it takes to cover a given area. This issue could be tackled by having two or more robots covering an area simultaneously, with factors like their battery level being taken into consideration at the time of deployment of assignment of area.

Another issue that did not fall within the scope of our project, but which is quite relevant, is the issue of obstacle avoidance during area covering. If the coordinates of the obstacles were known (such as might be in the case in a department store –clothing racks are located in specific positions that the vacuum cleaners should not cover), these coordinates could be fed to the master controller, which would assign areas around the obstacles. If the obstacles are not stationary or known beforehand, such as might be the case in robots deployed in a search and rescue mission, additional sensors and feedback mechanisms would need to be added to the slave robots for the purpose of avoiding these unexpected obstacles.

Overall, we are very pleased with the end result of our work. We were proud to see our project come together in the end, and we feel that our system provides a solid basis for many real-world applications to stem from.

For more information about this project, please visit our blog at http://falcopunch.blogspot.com/and the Github page for the project at https://github.com/yizhe/Multi-Robo-Cooperation