Model-Driven Configuration of Automated Parking Facilities

Abstract

This project represents an investigation into the customization of an environment that supports a fleet of autonomous vehicles cooperating to solve a common task. Specifically, the project is scoped within the context of an automated parking facility, whereby a driver may drop a car off at the entrance to a garage, receive a reservation code, and leave the car behind. The vehicle will then be instructed on how to drive itself to an open parking space. The owner of the car may return to the facility to retrieve their vehicle, which will be autonomously returned to the owner amid a set of other vehicles that may be entering and leaving the facility concurrently. Determination of parking location and maneuvering to and from the space is coordinated between each vehicle and a host controller at the facility, which can be configured for specific parking lots using a high-level modeling language. The controller handles all communication between vehicles and provides instructions to each car regarding the directions to the assigned parking space. For the purposes of this project, small robots were used to simulate cars and Bluetooth was the wireless communications medium between the cars and host controller.

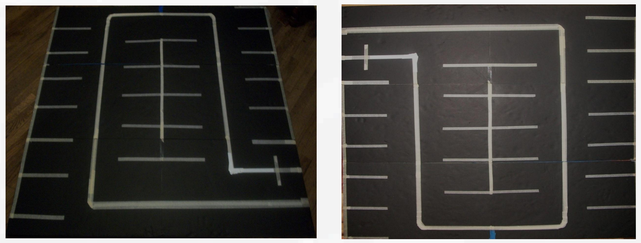
Purpose

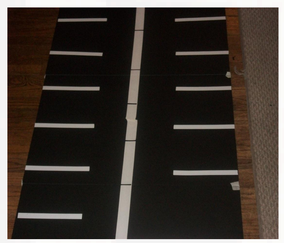
Our research focuses on the development of a smart garage that contains information about parking space availability and that is able to transmit that information to vehicles. The project provides an investigation that addresses this problem through a software-driven solution that supports customization through visual, domain-specific models.

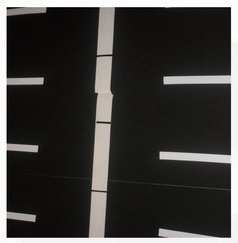
Objectives

Specific goals for this project were as follows:

- Investigating issues related to autonomous control of vehicles using the Lego Mindstorms NXT as an experimental platform.
- Designing an efficient method of traversal in a parking garage.
- Designing and implementing a communications protocol for the garage vehicle system.
- Implementing collision avoidance for vehicles moving in the garage.
- Developing a flexible, domain-specific modeling language for the customization of different parking garages, to allow for wider use in garage construction or improvement.







Foam board garage construction, initial and revised

These are the types of garages we simulated and made automated.

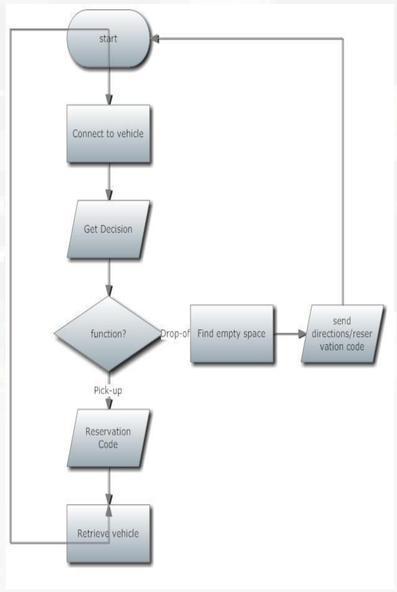
Methodology

Robotics and Communications

- We used Java as the programming language for this project..
- A parking garage and a car, represented by the NXT robot, are the main objects.
- A garage can be divided into different floors, or parking lots. These lots can be further subdivided into a basic unit, the parking space.
- The project was modularized into two parts, a component dealing with the robotics and communication between the robot and the garage, and the composition of the garage itself.
- From this, four classes (Garage, ParkingSpace, ParkingLot, and Car) were implemented.
- The host computer sends commands through Bluetooth to the robot using a string of characters.
- Each robot runs the same basic application. This client software contains:
  - collision avoidance algorithms
  - a line tracer algorithm for following the guideline
  - communication capabilities to retrieve the parking direction string from the host controller.

As vehicles enter the parking facility, they wait for a connection from the host controller in the garage, and then accept instructions from the host that represent navigation control and directions to a parking space. The flowchart depicts the flow of the autonomous navigation.

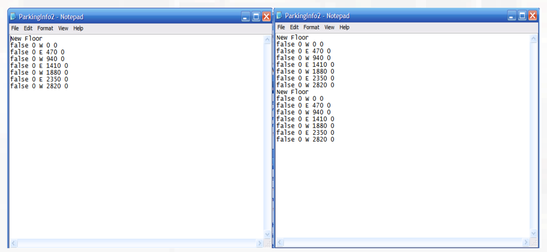
- We constructed the representative garage out of foam board. We delineated parking spaces using tape.
- We also used tape as the guideline.
- Initially, the robots would only use their tachocounts to find parking spaces, but parking was not always accurate.
- We revised our method to use a color sensor to detect parking spaces. This did not work as well as planned, due to the sensitivity of the color sensor.
- We then used the light sensor in place of the color sensor, and replaced the colored markers with black markers, as they were of a different light sensor reading than the foam board.
- This modification was more effective than using the color sensor, with the robots parking more accurately.



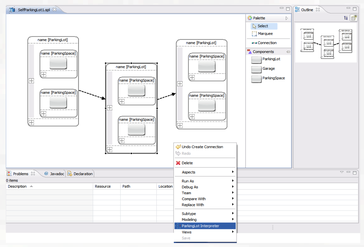
Visual representation of parking process

Domain-Specific Modeling

- The creation of the modeling tool can be broken down into two steps: creating the metamodel, which defines the domain-specific modeling language, and programming the interpreter.
- A Garage contains ParkingLots, and a ParkingLot contains ParkingSpaces. ParkingLots mimic the different floors of a garage. We developed a metamodel that conformed to the rules above, using the GEMS (Generic Eclipse Modeling System) plugin for the Eclipse IDE (Integrated Development Environment).
- Interpreters can be used to generate actual code in any programming language at the click of a button. In our case, we decided to use the interpreter to generate the attributes for each parking space in the garage in a text file.
- Another program was written that would read in this “space information” using file streams, and use this data to help a vehicle navigate through the parking facility.



Text files generated by the interpreter for two separate garage configurations. Each line in the text file represents a unique space and lists each of that space’s attributes.



Screenshot of actual modeling tool, and demonstration of how to invoke the interpreter in order to generate the space information.

- The Garage class keeps track of all the empty parking spaces in the garage using an ArrayList of empty parking spaces. Initially, all parking spaces start empty, filling up as cars enter and leave the garage.
- The garage updates the status of a parking space with each delivery and retrieval to and from that space.
- When a user wants to leave their vehicle, the garage connects to it through Bluetooth, and then send the appropriate data string.
- It will then fill the available space and store the Bluetooth address of the vehicle. When leaving the garage, the garage scans each space for the correct vehicle. Once found, the garage gives instructions on how to return to the start of the garage.
- The garage regenerates the reservation code for each space that is emptied.

Software Excerpts

```
public void stateChanged(SensorPort port, int oldValue, int newValue)
{

   if (sensor.readValue()>= lineValue)
    onLine = true;
   else
    onLine= false;
}

 public boolean getStatus()
{return onLine;}
```

This segment shows how monitoring of the Light(Color) sensor is done

```
//sweep to find line
   while (!getStatus()) {
     navigator.stop();
     navigator.rotate(sweep, true);
        while (!getStatus() && navigator.isMoving())
          Thread.yield();
        sweep *= -2;
```

This segment is a line following algorithm taken from a LeJOS sample.

```
public ArrayList<ParkingSpace> getAvailableSpaces () // get the empty spaces in the garage
{
  ArrayList<ParkingSpace> spaces = new ArrayList<ParkingSpace>();

  for (ParkingLot lot: lots)          //traverse each floor
  {
    ArrayList<ParkingSpace> floorSpaces = lot.getSpaces();
    for (ParkingSpace space: floorSpaces)  //traverse each space(in each floor)
    {
      if (space.isEmpty())
        spaces.add(space);
    }
  }
  return spaces;
}
```

The garage gets a list of available spaces by checking which ones are not filled.

```
public ParkingSpace(int l, int w, String direction, int count, int num)
{
  length = l;
  width = w;
  opening = direction;
  tachoCount = count;
  spaceNumber = num;
}
```

This shows how a parking space is constructed.

```
while(count < spacecount)
{

  while (sonar.getDistance() < threshold)
  {
    Motor.A.stop();
    Motor.B.stop();
  }

 Motor.A.forward();
 Motor.B.forward();
```

This code shows that the robot moves until its tachocount (or marker count) is equal to the tachocount (marker number) for the assigned space. The robot also only moves when it has a certain clearance.

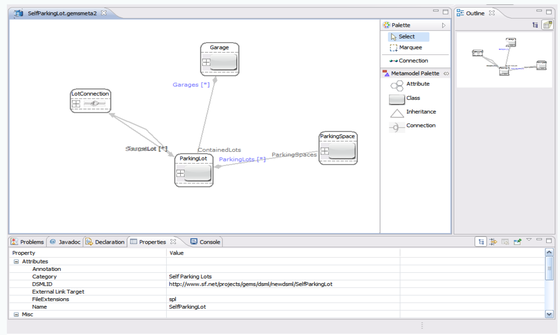
```
try {
   out.writeChar(movements.charAt(0));
   out.flush();

   movements = movements.substring(1);
   int num = Integer.parseInt(movements);

   out.writeInt(num);
   out.flush();
   out.close();
}
catch (Exception e)
{

}
```

This excerpt shows how the garage sends instructions to the NXTs. The first character sent is the direction the robot should turn into the parking space. The remaining is either the tachocount of the parking space, or the number of the parking space.



Screenshot of the metamodel. Connections between classes are containment relationships. The connection object (LotConnection) specifies which classes can be connected in the actual modeling tool. In this case, ParkingLots can be connected to other ParkingLots, mimicking consecutive floors in a garage.

```
/**
 * @param args
 * @throws FileNotFoundException
 */
public static void main(String[] args) throws FileNotFoundException {
    // TODO Auto-generated method stub
    int lengthSpace = 5;
    int widthSpace = 5; //no idea what the actual parking space length and width are, but
    //I just put in values because I needed them for constructor
    int floorCount = 1;
    Scanner in = new Scanner(new File("C:\\ParkingInfo2.txt"));
    ArrayList<String> strings = new ArrayList<String>();
    ArrayList<ParkingSpace> spaces = new ArrayList<ParkingSpace>();
    ArrayList<ParkingLot> lots = new ArrayList<ParkingLot>();
    boolean pastFirstFloor = false;

    while(in.hasNextLine()) //just reading in contents of file line by line
    {
        strings.add(in.nextLine());
    }

    StringTokenizer tokenizer; //tokenizer for each string in arraylist strings
    for (String s: strings)
    {

        if (s.equalsIgnoreCase("New Floor") && pastFirstFloor) //when there is a new floor
```

Above is an excerpt of code from the program that reads in parking space information from the text file generated by the interpreter. Data is read in line by line, so that each space is initialized with the correct attribute values.

```
public void visitParkingSpace(ParkingSpace toVisit) {
    // TODO Auto-generated method stub
    System.out.println("Visited ParkingSpace");
    String opening = "";
    int currTachCount = 0;
    int gapValue = 1210; //tachometer count to traverse part of lot with no spaces
    int avgTachCount = 470; //average tachometer count of space
    if (currSpace == 9) //in the parking lot layout, space 9 is special case - still faces
    {
        currTachCount = (currTachCount - 1) * avgTachCount + gapValue;
        opening = "W";
        currSpace++;
    }
    else if (currSpace > 9)
    {
        currTachCount = (currSpace - 9) * avgTachCount + gapValue;
        if (currSpace % 2 == 0)
            opening = "W";
        else
            opening = "E";
        currSpace++;
    }
    else
    {
        currTachCount = currSpace * avgTachCount;
        if (currSpace % 2 == 0)
```

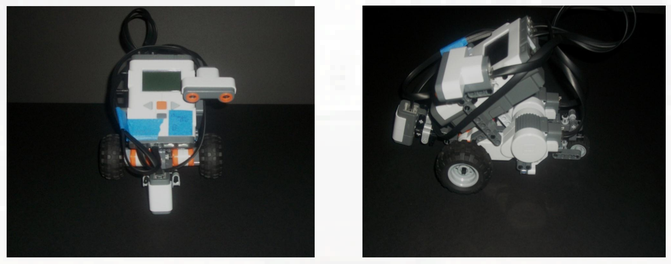
Above is a screenshot of the interpreter. This specific method, visitParkingSpace(), illustrates the process that generates code for each space in the garage.

The modeling tool greatly increases the versatility and adaptability of the garage. Since one is no longer confined to one fixed configuration of parking spaces, there are nearly limitless possibilities for different garage designs. While the use of domain-specific modeling has shown great promise, there is still much that needs to be addressed.

Results

The final version of the application achieved the goals of the project: the brain of the garage, the controller residing at the parking facility, is able to send Bluetooth instructions to multiple NXT robots concurrently to coordinate entry and exit of the facility. The robots are then able to parse that input and follow the instructions to park in specific parking spaces. The robots are also able to receive instructions to back out of parking spaces and exit the garage. Several video demonstrations have been captured and are archived on a project web site.

The research done in this project has shown that domain-specific modeling can be a very useful tool in the parking garage. With the focus on writing code diminished, it is now possible to focus on improving the functionality of the garage. However, there are still many limitations that have not been addressed in the project in terms of modeling. Our own inexperience with domain-specific modeling may have led to an inefficient design of the metamodel, and consequently, the modeling tool and interpreter. As of right now, however, the initial goals we set out to accomplish have been met. The modeling tool for the parking facility follows all rules laid out in the metamodel, and the code generator was able to write out the attributes for each parking space (such as tachometer counts) to a text file using file streams. This was accomplished for one physical garage and one other configuration.



Robot construction

Future Work

We were able to meet our initial goals that were enumerated at the start of the project. Multiple robots were able to navigate through the garage via Bluetooth instructions from a host controller, and the model can successfully generate multiple parking garage configurations. We were also able to implement collision detection and prevention within the garage using ultrasonic sensors that came with the NXTs. This enabled us to coordinate entry and exit of the parking facility with multiple vehicles. However, while the project has shown many promising signs, there is still much we can improve.

In the future, we will improve and add to the functionality of the project in the following ways:

- Improving the algorithm used to trace the guideline. This will ensure accurate positioning which will in turn lead to better parking.
- Adding sensors to parking spaces for extra confirmation and safety. The garage will know if some vehicle accidently went to the wrong space, and will therefore not send another vehicle to that space.
- Implementing some form of localization, such as Monte Carlo localization, to allow more accurate traversal of the parking garage. This could also improve safety in the garage.
- Implementing subsumption and more behavior-based programming. A robot follows a simple command, such as moving forward, but switches to more complicated procedures when obstacles are encountered, such as nearing a wall.
- Redesigning the robots so that they resemble traditional vehicles. We can then determine the best way to implement parking procedures for different types of automobiles.
- Implementing a method to quickly retrieve vehicles without scanning each parking space.
- Testing alternative implementations of the interpreter with other garage configurations.
- Developing a new metamodel that can make the modeling tool more intuitive.

The Lexus LS460 employs Lane Keep Assist (LKA) which helps drivers maintain position on the road. The car uses a stereo camera and other sensors to monitor position. This is similar to our concept of the NXTs following a guideline through the garage.