Robot Code Documentation for Project 3
April 30, 2003
Team 4 – Justin Fuller, Rahul Kotamaraju, Matthew Lawrence

1.0 Overview

The software used for the third project was designed to incrementally incorporate the robot's sensor array and accomplish the tasks necessary for efficient completion of the assignment. The architecture is almost completely deliberative, but it does include a small amount of reactive behavior in the alignment routines. The main algorithm consists of three major tasks: obtain the closest block, move to the closest destination location, and drop off the block. The robot repeats these three tasks until it obtains and delivers all known blocks. Additionally, the software structure allows for a world model that can be arbitrarily modified before runtime.

2.0 World Model

The team designed the world model after determining which facets of the project environment are necessary for the robot to function properly. The world model includes four major categories: block position, destination position, block population, and robot position. Though the code does not explicitly store it in any data structure, the code assumes an arena representation of an 8 foot by 12 foot grid with resolution of one half foot. There is a special function, *acceptInputWM*, the purpose of which is to incorporate user-defined input values into the operational world model.

2.1 Block Position

There may be anywhere from zero to sixteen blocks in the arena during any given run. The world model allows for an x and y coordinate for each of sixteen blocks. The world model assumes every orange block is a point. For those blocks that don't exist, negative 1s are placed in the model input, as specified by the instructor, Dr. Hougen. The robot replaces these negative 1s with 9999s, essentially erasing the block from the world model by making it infinitely far away.

2.2 Destination Position

The arena may contain as many as four destination locations, but no less than one. As with blocks, the world model designates each destination location as a single point, which represents its center. The six-inch margin, which black electrical tape demarcates, is assumed in robot operation. However, there is no allowance for this margin in code.

2.3 Block Population

The code determines how many blocks are present in the world based on input at design time. This value is used for comparison with a counter that is incremented each time a block is delivered; in this way, the robot knows when it has delivered the expected number of blocks.

2.4 Robot Position

The robot's position is a necessary item for determining the best way to interact with blocks and destinations. A combination of location and direction, the robot constantly updates its own position when the *navigate* function is called (see code).

3.0 Tasks

The robot accomplishes three complex tasks using the world model. Though all of them share particular functions, such as *navigate* and *cage*, they are very different conceptually.

3.1 Obtain Nearest Block

The robot uses its world model to find out which of the sixteen possible blocks is supposedly closest (the *findClosestBlock* function). This involves several distance calculations, and the robot is not moving when it performs these computations. Once a candidate is selected, the robot first “erases” the block from the world model by giving it impossibly large coordinates, and then it *navigate*s to the block location from its current position. In order to account for a possible dislocation of about one half foot, the robot then calls the *CorrectError* function and *navigate*s forward an additional five inches to ensure that the block is in the cage. The robot then locks the cage in preparation for the next step.

3.2 Move to the Closest Destination Location

The robot again uses its world model to find out which destination location is closest to its current position (using *findClosestDest*). After determining the best destination, the robot *navigate*s there, recording its movement. Once the action is complete, the cage is placed in the release position.

3.3 Drop Off the Block

After the block is released, the robot performs *alignAxle*, which uses two independent threads that poll the IR sensors. The robot responds to black tape in reactive fashion to make slight adjustments to the vehicle’s orientation. The robot then determines if it is in the north or south side of the arena, and navigates a safe distance toward the other side before signaling delivery and opening its cage.

4.0 Conclusion

Though there are many aspects of the code that are quite complex, the robot still errs on the side of simplicity. There are many factors that it doesn’t take into account, and this leads to a lack of efficiency. However, even in these cases, the robot reacts reasonably and should make a fairly high score due to its redundancy and resilience to unexpected conditions.