Project 3 Software Documentation

2.1 – Introduction

The software design that was chosen by our software development team was similar to the same software design that was used with our second project. This design was to use a finite state machine as the top layer, but unlike the previous software, there is only one state machine instead of many state machines inside another. One reason for this new design was for simplicity of the system. Our last project became hard to read and follow if you were not familiar with its design. A second reason is that this project was less complicated in the way of states and behaviors the robot had to deal with. This together allowed us to develop an efficient and modular system which we will explain in more detail below.

2.2 – World Model

Our software uses a world model in order to plan its path to the blocks and the goal coordinates. Our first design was to use an two dimensional array and plot all the block and goal points as well as track the robots location through the world as it moves around. However we found this to be very cumbersome and heavy on the system in terms of memory usage. We instead opted to use a more dynamic world model by using the coordinates given to us at the start of the run to keep track of the goals and the blocks since the coordinates were set up in a standard format. Also instead of moving our robot location through the two dimensional world model we decided to dynamically update a pair of coordinates as the robot moved according to its 1) odometer, 2) compass heading, and 3) direction of turn. This proved to be just as effective and efficient in allowing the robot to know where it is in relation to all of the blocks and goals. With this world model scheme, path planning was also very easy to calculate and execute.

2.3 – Path Planning

Path planning was kept to the simplest form possible in order to minimize the amount of error the robot would encounter with navigating turns and keeping track of its heading. We used the distance formula to calculate the distance to all of the blocks in the world model array. Keeping track of the minimum distance coordinates as we went along we would then use those coordinates along with the robots coordinates to plan a path that would use just 1 90 degree turn to get to the goal destination. The robot would calculate the number of plus or minus x moves it would need to be on the same x plane as the goal and then turn in the direction of the goal and move forward plus or minus y moves to arrive at the goal. The path planed was not the shortest path but was the most efficient in calculating with minimum error since the robot did not need to calculate any type of degree to turn itself into the direction of the block or goal. Upon completion of the path planning it then passed the plan created to the plan executor do carry out the set of instructions.

2.4 – Plan Execution

The plan executor looks at a global array that was filled in by the path planner with a series of steps to take. The steps are pretty simple and include things such as move forward, turn right, turn left, backwards, etc. Each of the move forward commands tells the robot to move forward 6 inches. The plan executor will look ahead in the array and sum up an entire string of forward commands and combine them into 1 long forward command as to allow the robot to move smoothly down a long straight away instead of stopping to look at each instruction to see if it must move forward again. When the planner receives and turn command it then turns and executes the next line of instructions. This is carried along until the planner receives a "PLAN_END" command which tells the planner there is no more commands to execute. The robot then evaluates its position to determine if it is in the right coordinate and then looks for the block if it is at its destination.

2.5 – Conclusion

Our software was very capable of planning and executing tasks for any type of coordinates and goals given to it. Our problem that we had with the robot was controlling the turns to be exactly 90 degrees. We tried to minimize the amount of turns the robot would need to take which did help but overall after a few turns the robot was off by a coordinate and then would miss the goal by a few inches and the more turns the robot took the more it was off. We did not have time to implement a correction algorithm to correct our coordinates on the goal as we did not feel it would help much if the robot missed the goal entirely. To improve on this we feel we should have spent more time trying to work the hardware to give us better encoder resolution and use that to correct the turns to a more exact degree.