Lab 6: LabVIEW iRobot Obstacle Avoidance and Hill Climb

Sam Mansfield and Toan Vuong
TA: Hoekun Kim
EECS149

October 16th, 2013

1 Introduction

2 Analysis

We didn't need to add any states from the obstacle avoidance FSM, but we added several guards. In addition to just detecting hills and climbing them we implemented left and right obstacle detection. Previously we always turned left to go around an object, but Hokeun insisted that we should add the right and left avoidance feature, which we did. Basically when the left sensors pick up an obstacle the obstacles are incremented, the robot turns right, and when back in the original orientation obstacles are decremented. On the other hand if the right sensors detect an obstacle, the obstacles are decremented, the robot turns left, and when back in the original orientation obstacles are incremented.

In order to climb hills, we added two guards. One detects if we are on a hill. In this case we turn the boolean orient to true and rotate until the x axis of the accelerometer is positive (meaning the robot is facing uphill) and the y axis is close to 0 (meaning that the robot is facing directly up the hill). In order to take into account for movement of the robot, we had the robot move very slowly, and it seemed to work quite well. Previously in last lab, we made sure the robot turned a certain amount in order to correct, but this actually caused a lot of problems in simulation.

Speaking of simulation problems, we had a tremendous amount of trouble in simulation. We got things working in the physical world for the hill climb pretty quickly since we had our algorithm from the previous lab, but we could not use that algorithm in the simulation. At first we thought the accelerometer values were not getting passed correctly to the state chart, but what it turned out to be was the net angle wrapping, which we weren't taking into account. Originally we wanted to have the robot turn at least 45 degrees before trying to detect the top of the hill, and in the real world this works fine, in order to take into account of the movement of the accelerometer. In simulation this did not work, the angle would wrap around so that when the guard was supposed to be true, the net angle had wrapped and now the guard was no longer true, so our robot was orienting continuously and never went up the hill.

The differences between simulation and the real world mainly had to do with magic numbers. For instance in the simulation we might use something like when z is less than 0.99, but in the real world when z is less than 0.9 is a more reasonable value. Another differences is that in the simulation 90 degrees means 90 degrees, but in the physical world 90 degrees is more like 73 net angle degrees. It only came down to magic numbers that make the robot transition smoother.

Considering how robust LabVIEW is, we were not able to easily debug our program, we couldn't figure out how the probe tool worked, and considering that in the real world we our algorithm worked it was intensely frustrating for the simulation to continuously not work. As much as LabView pretends to be a friendly environment it is much easier for us to write and debug the robot in C than it was in LabVIEW.

3 Conclusion

In this lab we learned how to use labVIEW state charts to design an algorithm to avoid obstacles and climb hills. We learned about the simulation capabilities and structure that can be used to create projects in LabVIEW.

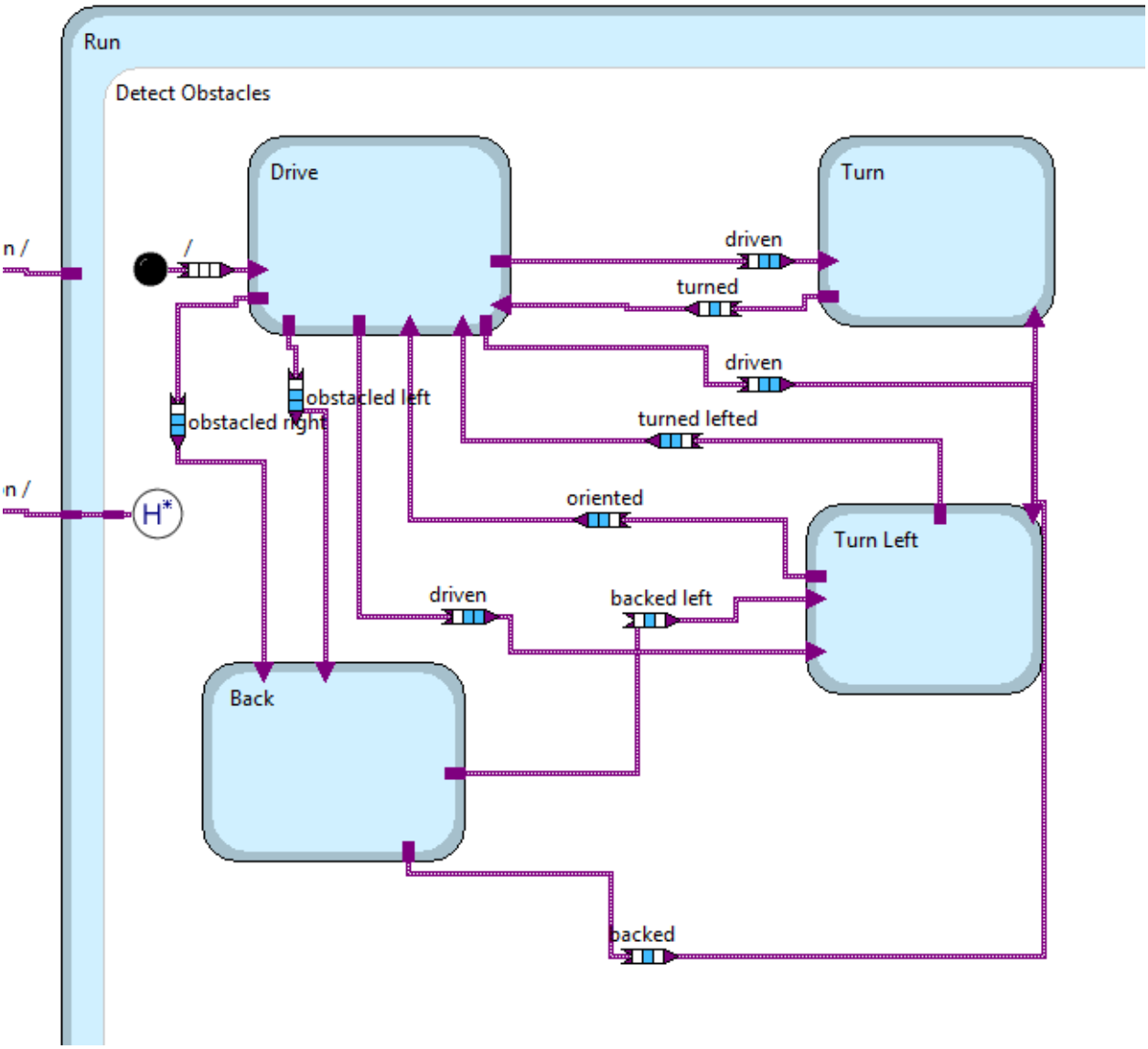

Figure 1: State Chart for obstacle avoidance and hill climb.