Lab 5: iRobot Hill Climb in C

Sam Mansfield and Toan Vuong
TA: Hoekun Kim
EECS149

October 9th, 2013

1 Introduction

In this lab we enhanced our previous navigation algorithm to allow the iRobot to navigate up a hill, while remaining aware of and going around obstacles. We made decisions about the priorities of various parts of our algorithm, how often to detect hills, and *what* exactly is a hikk given the robot's sensors.

2 Analysis

Our navigation algorithm remains the same from the last lab. In addition, we also occasionally check for hills by utilizing the myRIO's accelerometer. On an incline, the robot's acceleration in the z-axis will be less than 1g, whereas the magnitude of acceleration in the x and y axes will be more than zero. Using this knowledge, we transition the robot from driving straight to turning when a specific threshold for each accelerometer reading is met.

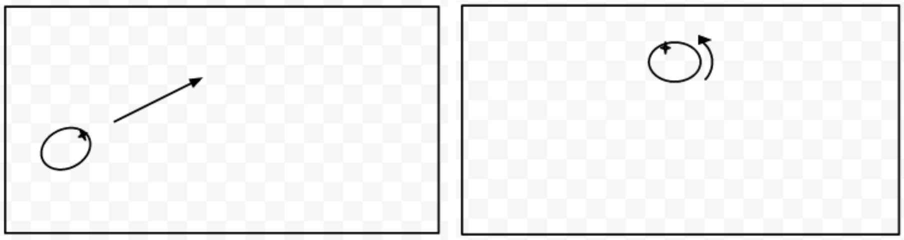


Figure 1: Bird's eye view: The robot making decisions about moving straight vs. turning on a hill.

The robot will continue to turn until its orientation is forwarding-facing up a hill. To determine the orientation of the robot, we again used accelerometers. Specifically, we know that when the robot is in this orientation, its x-axis reading should be positive, whereas its y-axis reading should be close to zero. This is posited from the way the accelrometers are mounted on the robot. When each sensors are outside of their respective thresholds, we have determined that the robot is now in the correct orientation to continue moving up the hill.

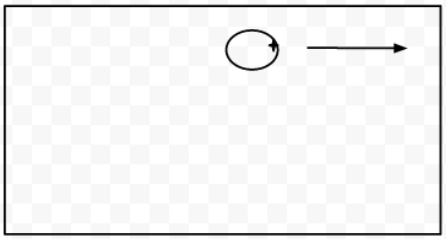
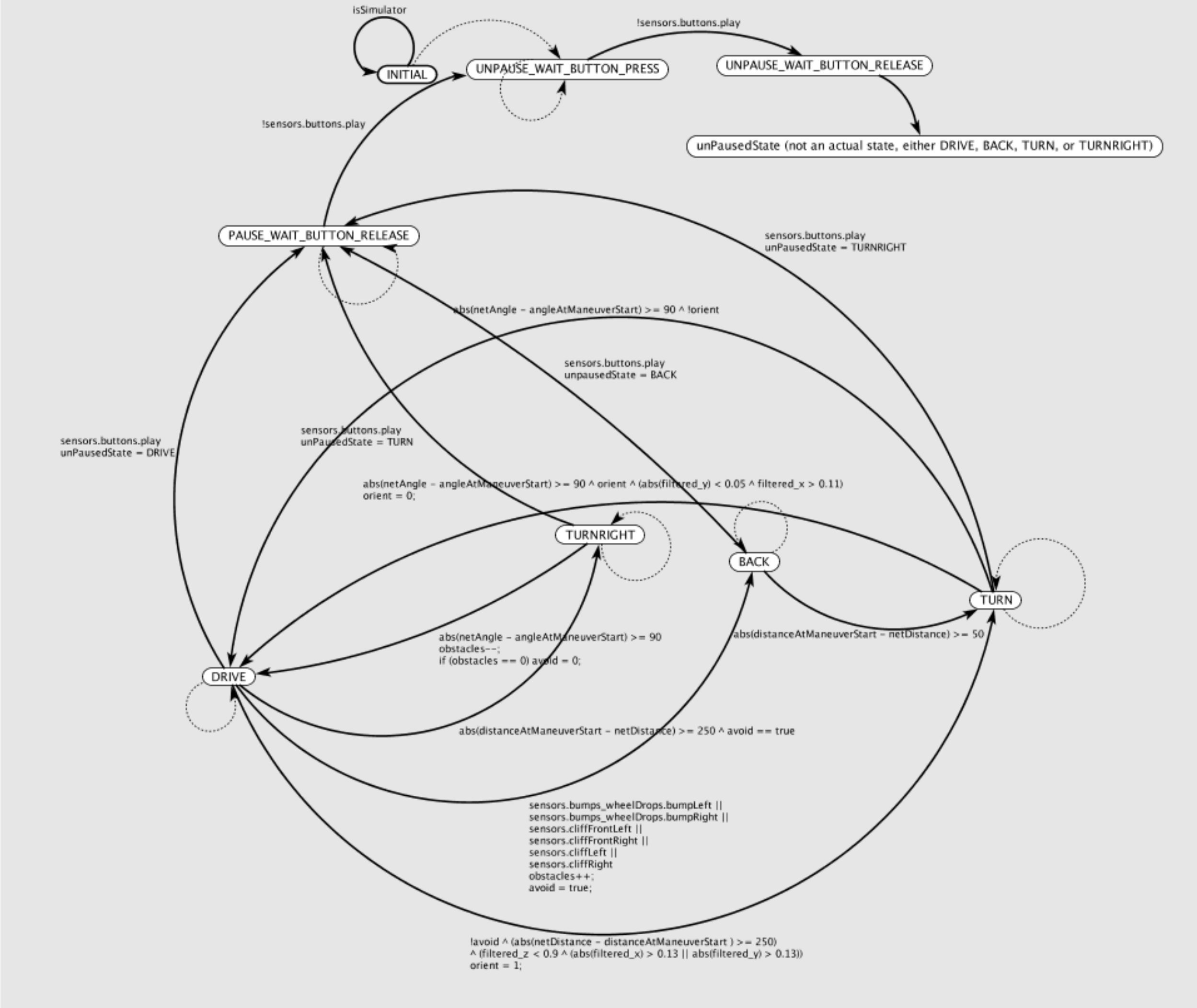


Figure 2: Bird's eye view: The robot resuming an upwards trajectory.

The last piece of our algorithm involves prioritizing obstacle avoidance versus hill climbing. We merely allowed obstacle avoidance to take precedence over hill climbing: That is, given that a robot is climbing a hill, it will begin its obstacle avoidance algorithm irrespective of which state of the hill climbing algorithm it is in. It will only resume hill climbing once there are no more obstacles. This is a necessary prioritization in order to prevent the robot from falling off a cliff in its quest up the hill.

To implement this algorithm we followed a state machine architecture without adding any new states from Lab 4. However, we did add additional guards and transitions.


isSimulator
INITIAL
UNPAUSE_WAIT_BUTTON_PRESS
!sensors.buttons.play
UNPAUSE_WAIT_BUTTON_RELEASE
unPausedState (not an actual state, either DRIVE, BACK, TURN, or TURNRIGHT)
!sensors.buttons.play
PAUSE_WAIT_BUTTON_RELEASE
sensors.buttons.play
unPausedState = TURNRIGHT
abs(netAngle – angleAtManeuverStart) >= 90 ^ !orient
sensors.buttons.play
unpausedState = BACK
sensors.buttons.play
unPausedState = DRIVE
sensors.buttons.play
unPausedState = TURN
abs(netAngle – angleAtManeuverStart) >= 90 ^ orient ^ (abs(filtered_y) < 0.05 ^ filtered_x > 0.11)
orient = 0;
TURNRIGHT
BACK
TURN
DRIVE
abs(netAngle – angleAtManeuverStart) >= 90
obstacles--;
if (obstacles == 0) avoid = 0;
abs(distanceAtManeuverStart – netDistance) >= 50
abs(distanceAtManeuverStart – netDistance) >= 250 ^ avoid == true
sensors.bumps_wheelDrops.bumpLeft ||
sensors.bumps_wheelDrops.bumpRight ||
sensors.cliffFrontLeft ||
sensors.cliffFrontRight ||
sensors.cliffLeft ||
sensors.cliffRight
obstacles++;
avoid = true;
!avoid ^ (abs(netDistance – distanceAtManeuverStart) >= 250)
^ (filtered_z < 0.9 ^ (abs(filtered_x) > 0.13 || abs(filtered_y) > 0.13))
orient = 1;

To account for movement we would only take measurements for tilt after the robot was either in mid drive or mid turn. By doing this the robot should be close to constant velocity and have little effect on the tilt sensing.

We had two problems in this lab. One was that we assumed that the accelerometer data being fed to us was already filtered, since in the comments it said it was filtered. After we added our own filter the data was much more stable. Another problem we encountered was that the `abs()` function only takes integers as arguments. We were feeding in doubles, which did not produce the desired result. We made our own `absDouble()` function and this improved our algorithm dramatically.

3 Conclusion

This lab helped us further our use for state machines and also helped us think about how to use accelerometer data effectively. We learned how powerful state machines can be when used properly and how easy it is to change a state machine to add more functionality without having to rewrite the old code, we didn't change our obstacle avoidance state machine at all, only added to it. Learning to use the accelerometer data to sense tilt was very helpful, that way you don't need additional hardware to sense tilt.