Nicolas Meneses
Embedded Computing Robotics

FSM Description

Inputs:
- uint8_t data (Data from sensor line)
- An eight-bit number to get a 'view' of the robot with respect to the line - uint8_t bumpData (Data from bump switches)
- Either 0 (no bump switch being pressed), greater than 0 (bump switch being pressed).
- float light: 4 byte number with light sensor LUX level readings
- float acc1: 4 byte number with accelerometer reading on the z-axis

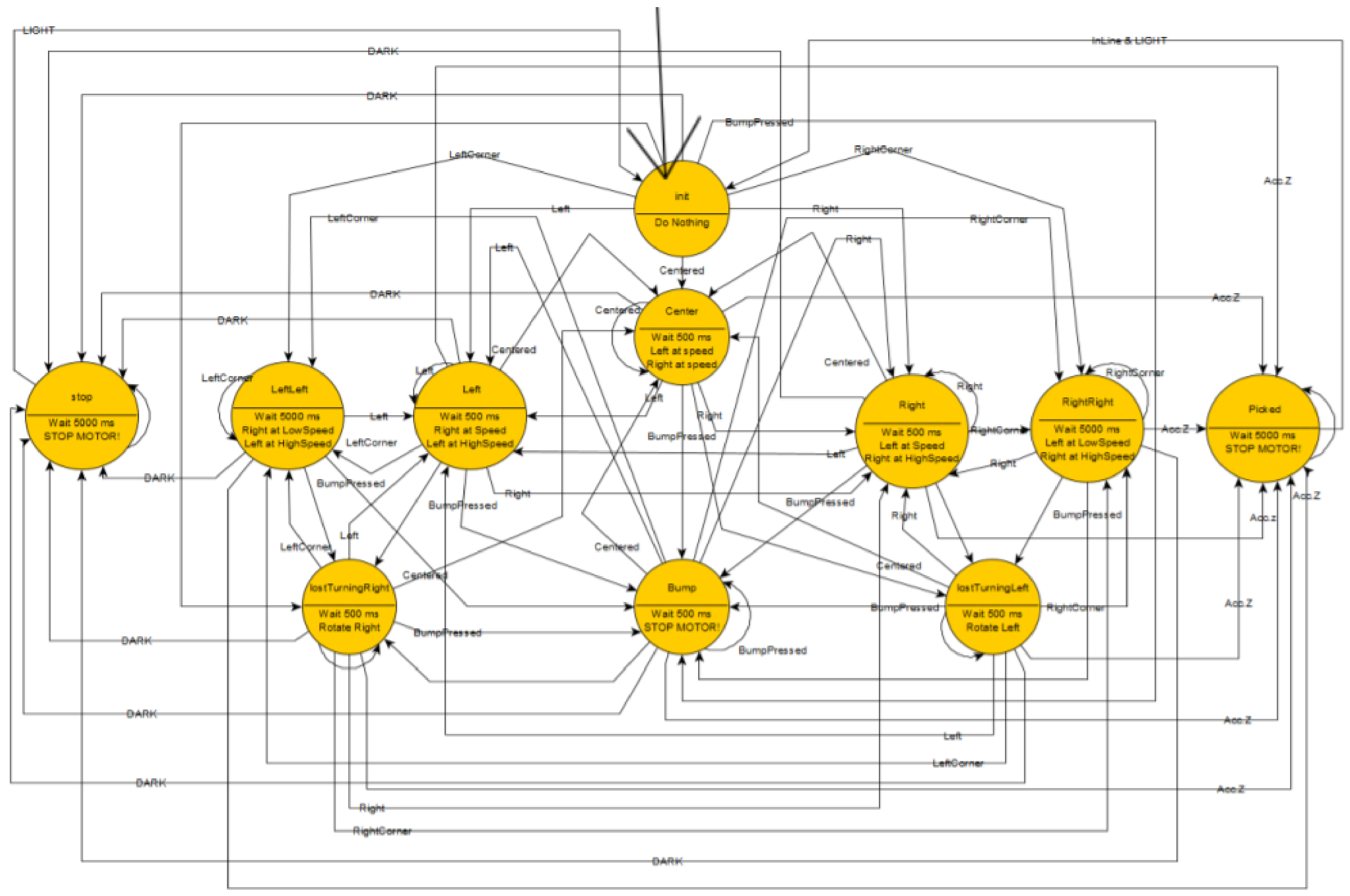
Outputs:
- Motor_Forward() - Function that drives both motors (left and right) forwards at a given duty cycle.
- Motor_Right() - Function that drives the right motor backward and the left motor forward at a given duty cycle.
- Motor_Left() - Function that drives the left motor backward and the right motor forward at a given duty cycle.
- Motor_Stop() - Function that stops the motors and enters sleep mode.

Input Labels for FSM:
- Centered: (data == 24 && bumpData==0)
- Left: ((data == 8 || data == 12 || data == 4) && bumpData==0)
- LeftCorner: ((data == 6 || data == 2 || data == 3 || data == 1 || data == 15 || data == 7 || data == 14) && bumpData==0)
- Right: ((data == 16 || data == 48 || data == 32) && bumpData==0)
- RightCorner: ((data == 96 || data == 64 || data == 192 || data == 128 || data == 240 || data == 112 || data == 224) && bumpData==0)
- BumpPressed: (bumpData>0)
- LIGHT: light>100
- DARK: light<=100
- Inline: data == bx11111111
- Acc.z: acc1>18500 || acc1<15000

Output Labels for FSM:
- Right at Speed: Right Motor drives forward at a 13% duty cycle
- Right at HighSpeed: Right Motor drives forward at a 20% duty cycle
- Right at LowSpeed: Right Motor drives forward at a 5% duty cycle
- Left at Speed: Left Motor drives forward at a 13% duty cycle
- Left at HighSpeed: Left Motor drives forward at a 20% duty cycle
- Left at LowSpeed: Left Motor drives forward at a 5% duty cycle
- Rotate Right: Right motor drives backward at a 13% duty cycle and Left motor drives forward at a 13% duty cycle.
- Rotate Left: Left motor drives backward at a 13% duty cycle and Right motor drives forward at a 13% duty cycle.

Nicolas Meneses
Embedded Computing Robotics

FSM:



Nicolas Meneses
Embedded Computing Robotics

Strategy & Policy

The behavior of the robot is controlled by 11 states. The stop state is reached by all other states as it makes the robot stop when ‘light’ is under 100. This state has the highest priority and is reachable from all other states. The bump state is reached by all other states as it makes the robot stop when it hits something. This state has the second highest priority and is reachable from all other states as we don’t want to burn the motors. The ‘picked’ state is reached by all other states (except init) as it makes the robot stop when it has been ‘picked’ up. This state has the third highest priority and is reachable from all other states as we want the robot to stop when the user picks it up. All of the other states and transitions among them occur exclusively with the “data” variable that comes from the line sensors (transitions only occur among states that cause movement when no bump switches are being pressed). With that being said, my approach for the states that make the robot move was to create a “center state” to make the robot move forward when centered in line. Then I created the “Left” and “Right” states to make the robot move slightly left or right, these states are reached when the bits 3 to 6 from the line sensor data change (in other words, when the robot is left or right from the lane but just slightly). Then I created the “RightRight” and “LeftLeft” states which make the robot turn rapidly either left or right, these states are only reachable from either the “Right”, “Left”, or “Lost” states and make the robot do a sharp curve after they have previously been making a turn; these states are reached when the bits 1-2 and 7-1 from the line sensor data change (in other words, when the robot is left or right from the lane by a considerable distance). Then I created LostTurningLeft and LostTurningRight which are the states reached when the robot does not detect the line. LostTurningLeft is reached after the robot is turning left and loses the line (from the “left” and “LeftLeft” states); this state makes the robot rotate left under the assumption that if it was previously turning left, the line can be found by rotating to that same direction. The same logic applies to LostTurningRight. Finally, the “init” state does not do anything and only sets the initial state of the Robot based on its initial position.