Week 8 Studio 1

A0214561M

Zhuang Jianning

Question 1

void leftISR()

{

if (dir == FORWARD) {

leftForwardTicks++;

forwardDist = (unsinged long) ((float) leftForwardTicks / COUNTS\_PER\_REV \* WHEEL\_CIRC);

}

else if (dir == BACKWARD) {

leftReverseTicks++;

reverseDist = (unsinged long) ((float) leftReverseTicks / COUNTS\_PER\_REV \* WHEEL\_CIRC);

}

else if (dir == LEFT) {

leftReverseTicksTurns++;

}

else if (dir == RIGHT) {

leftForwardTicksTurns++;

}

}

void rightISR()

{

if (dir == FORWARD) {

rightForwardTicks++;

}

else if (dir == BACKWARD) {

rightReverseTicks++;

}

else if (dir == LEFT) {

rightForwardTicksTurns++;

}

else if (dir == RIGHT) {

rightReverseTicksTurns++;

}

}

When dir is FORWARD, both wheels rotate forward so we increment leftForwardTicks (in leftISR) and rightForwardTicks (in rightISR). When dir is BACKWARD, both wheels rotate backward so we increment leftReverseTicks (in leftISR) and rightReverseTicks (in rightISR). We keep track of these 4 variables to differentiate which ticks were triggered moving forward or backward.

When updating forwardDist and reverseDist, we assume the number of clicks by the left and right encoders is similar, hence we only need to update both distances in leftISR using leftForwardTicks or leftReverseTicks. There is no need to update both distances in rightISR as the values should be the same.

When dir is LEFT, the left wheel rotates backward while the right wheel rotates forward, effectively allowing Alex to pivot left on the spot (like a tank) so we increment leftReverseTicksTurns (in leftISR ) and rightForwardTicksTurns (in rightISR). When dir is RIGHT, the left wheel rotates forward while the right wheel rotates backward, effectively allowing Alex to pivot right on the spot so we increment leftForwardTicksTurns (in leftISR ) and rightReverseTicksTurns (in rightISR). We keep track of these 4 variables when Alex is turning.

We do not update forwardDist and reverseDist when Alex is turning left or right as Alex is only pivoting on the spot in the left or right direction, hence no distance is covered.

Question 2

When sending the commands to move forward, reverse, turn left and turn right, we first set the packetType field to become PACKET\_TYPE\_COMMAND. Based on the command input, we set the command field to the corresponding enum constant such as COMMAND\_FORWARD, COMMAND\_REVERSE, COMMAND\_TURN\_LEFT, or COMMAND\_TURN\_RIGHT. We also get user input through the getParams function where we scan in the distance in cm/angle in degrees to param[0] and the speed/power in percentage to param[1]. We then send this command packet to the Arduino.

Question 3

The first parameter corresponds to the data we sent stored in param[0] of the command packet. For forward and reverse, it would be the distance in cm Alex would travel forward or backward. For left and right, it would be the angle in degrees Alex woud turn left or right. However, since the first parameter is not assigned to any variable in the movement functions, Alex would move forward, backward or turn left and right indefinitely.

The second parameter corresponds to the data we sent stored in param[1] of he command packet. For all 4 functions, it is the speed expressed in percentage which the motors will turn. The speed in percentage is converted to an analog value between 0 and 255 and used in analogWrite to drive the corresponsding motors. The higher the percentage, the faster the motor will turn. The lower the percentage, the slower the motor will turn.

Question 4

sendOK() creates a packet of packetType PACKET\_TYPE\_RESPONSE and command RESP\_OK, then calls sendResponse(). sendResponse() serialises the okPacket and sends it to the Raspberry Pi over the serial port. The receiver thread at the Raspberry Pi receives the packet, deserialises it and handles the packet if PACKET\_OK. Our okPacket of PACKET\_TYPE\_RESPONSE then calls handleResponse and prints out “Command OK” if the command field is RESP\_OK. This is important to ensure the command we sent to the Arduino has been processed correctly and returns an OK response, else we call sendBadCommand or any other error which informs us of the warning on the Raspberry Pi/User Interface.

Question 5

case COMMAND\_TURN\_LEFT:

sendOK();

left((float) command->params[0], (float) command->params[1]);

break;