Matt Donnelly

Daniel Bjorklund

Link for video: https://www.youtube.com/watch?v=zxSfUf110G0&feature=youtu.be

Lab 4 report

1a. We found this first part to be easy, as we already know how to use and make a display task. The only new part we had to get to know was the touch sensor. We were able to guess (before we did the reading) that the sensor would be using a Boolean to tell us the status of the button. We decided to have the sensor and display running always on separate tasks. The sensor sets a variable bumps equal to whatever the value of the touch sensor is. Bumps is reset after the value is greater than 2. We did this, so we know that the sensor was bumped and not being pressed. We also have a value touch which does the same thing, but that does not reset the value. In the display task we check to see if touch has a value, if so the button is pressed. Else if bump and not touch then we know the button has been bumped. The last resort means the button is released.

```
task sensor(){
    while (true){
        bumps=getBumpedValue(touchSensor);
        if(bumps>2){
            resetBumpedValue(touchSensor);
            bumps = 0;
        }
        touch=getTouchValue(touchSensor);
    }
}
```

```
task display(){
    while (true){
        if(touch){
            displayTextLine(10, "Pressed");
            }else{
            if(bumps && !touch){
                displayTextLine(10, "Bumped");
            }
            else{
                displayTextLine(10, "Released");
            }
        }
    }
}
```

1b. We found this question to be simple, just a matter of putting it together. We check the value of the touch sensor, all the time as it is in the while loop. If the value is one (or basically anything but zero), then we call turn and reset the value to zero. Turn then turns the vehicle 360 degrees. We have ratio (direction) which determines the direction of the next spin, by changing the sign. Each time the button is hit, the ratio is changed setting the turn direction for the next time the button is hit.

```
void turn(){
    setMotorSyncEncoder(leftMotor, rightMotor, ratio,825, 100);
    ratio = ratio*-1;
}
```

```
task main(){
    while(true){
        if(getBumpedValue(touchSensor)){
            turn();
            resetBumpedValue(touchSensor);
        }
    }
}
```

1c. We use the same method above, but this time we changed the main to a task as didBump. So the robot will check to see if the robot is moving. If the bump sensor is hit, and the motors are moving, the motors will stop. We also stop the turn task along with the motors, this is to ensure everything has stopped. So, when the sensor is pressed, the robot checks to see of the wheels are spinning, if so then it will stop the motors. If the motors are not spinning, then it will start the motors. The rest was taken from the first parts of the question.

```
task didBump(){
    while (true){
        if(isMove){
            if(getBumpedValue(touchSensor)){
                setMotorSync(leftMotor, rightMotor, 0, 0);
                resetBumpedValue(touchSensor);
                stopTask(turn);
            }
        }
    }
}
```

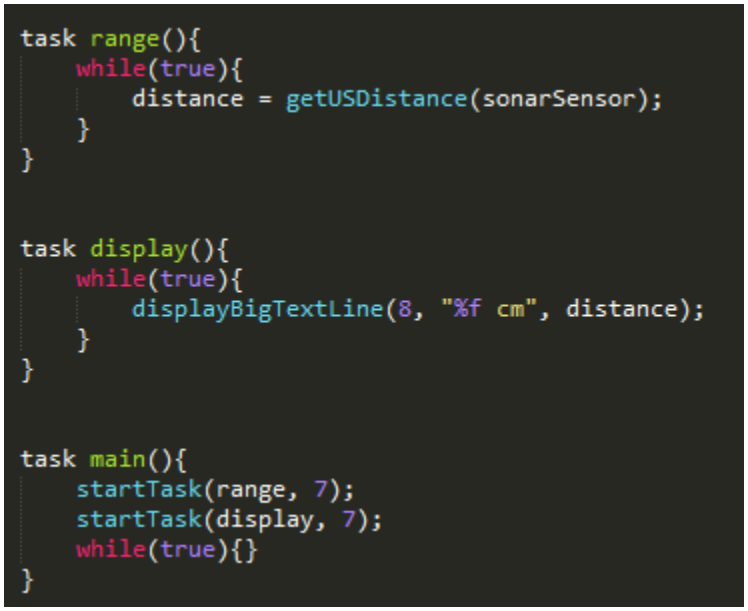
```
task isMoving(){
    while(true){
        if(getMotorRPM(leftMotor)!=0 || getMotorRPM(rightMotor)!=0){
            isMove = true;
        }
        else{
            isMove = false;
        }
    }
}
```

```
task main()
{
    startTask(isMoving, 7);
    startTask(didBump, 7);
    while(true){
        if(getBumpedValue(touchSensor) && !isMove){
            resetBumpedValue(touchSensor);
            startTask(turn, 7);


        }
    }


}
```

2a. This was simple as we had a display task running which took a global variable from range and displayed it on the LCD. We used getUSDistance to figure out how far an object is, and that was assigned to distance. We have the display running on a separate task, so the LCD displays live update of the range. The main task just starts the other tasks and keeps running.



```
task range(){
    while(true){
        distance = getUSDistance(sonarSensor);
    }
}


task display(){
    while(true){
        displayBigTextLine(8, "%f cm", distance);
    }
}


task main(){
    startTask(range, 7);
    startTask(display, 7);
    while(true){}
}
```

2b. Once again, we have a display task which displays the current distance from the nearest object. The display is a separate task so that it can always be updated/live feed. The distance is a global variable which is assigned in the range task. Again, this is a separate task, so it will always update, therefor updating the display. The range task just assigns the distance from the sensor. Main task checks the distance, then if it is less than 60 mm away, it will call the new controller task else the power is 0. The controller task checks the distance and the and the error. It then determines if the robot needs to go forward or backward, keeping the robot 30 mm away.

```
task display(){
    while(true){
        displayBigTextLine(8, "%f cm", currentDist);
    }
}
```

```
task main(){
    startTask(display, 7);
    startTask(range, 7);
    startTask(motorControl, 7);

    while(true){
        if(currentDist<60){
            startTask(newController, 7);
            sleep(100);
        }
        else{
            currentPower=0;
        }
    }
}
```

```
task motorControl(){
    while(true){
        setMotorSpeed(leftMotor, currentPower);
        setMotorSpeed(rightMotor, currentPower);
    }
}
```

```
task newController(){
    error = desiredDist - currentDist;
    if(abs(error)>tolerance){//this should take care of
        desiredPower = k*error*-1;//this will be negativ
    }else{
        desiredPower=0;
    }
    if(error<0){//error calculation already tells us whi
        if((currentPower + slewRate) < desiredPower){
            currentPower += slewRate;
        }else{
            currentPower = desiredPower;
        }if (currentPower> maxPower){
            currentPower = maxPower;
        }
    }else{//the robot must move backward
        if((currentPower - slewRate) > desiredPower){
            currentPower -= slewRate;
        }else{
            currentPower = desiredPower;
        }if (currentPower< -maxPower){
            currentPower = -maxPower;
        }
    }
}
```

```
task range(){
    while(true){
        currentDist = getUSDistance(sonarSensor);
    }
}
```