Matt Donnelly
Daniel Bjorklund

Lab 3 report

1. This program relies on our slew rate program from lab 2 that we created. Using that as a base, all we added was a way to calculate how fast it was going (done by the convertRad function) and a function that acts as a controller for our slew rate (angularVelControl). This simply checks if we are above, below, or at the desired speed and then calls motorspeed (our slew rate function) with the correct arguments for the situation. It supports the individual motor speeds through the two tasks motorControlTaskL/R.

```
#pragma config(StandardModel, "EV3_REMBOT");
float tolerance;
int increment;
int desiredPowerL;
int desiredPowerR;
int currentPowerR;
int currentPowerL;

float desiredSpeedL;
float desiredSpeedR;
float currentSpeedL;
float currentSpeedR;

int powerIncrement;



float convertRad(float x, tMotor motor){


    return ((x*(2*PI))/60)*sgn(getMotorSpeed(motor));
}

void motorspeed(int desiredPower, int *currentPower){
        while (*currentPower != desiredPower){
            if (desiredPower!=*currentPower){
                if (*currentPower < desiredPower){
                    if(*currentPower + increment > desiredPower){
                        *currentPower = desiredPower;
                    }
                    else{
                        *currentPower += increment;
                    }
                }
                else{
                    if(*currentPower - increment > desiredPower){
                        *currentPower = desiredPower;
                    }
                    else{
                        *currentPower -= increment;
                    }
                }
            }
            sleep(100);
        }
    }
```

```
void angularVelControl(float *desiredSpeed, float *currentSpeed, tMotor motor, int *desiredPower, int *currentPower){

    while(true){
        *currentSpeed = convertRad(getMotorRPM(motor), motor);


        if (abs(*desiredSpeed - *currentSpeed) > tolerance){
            // need to increase and decrease via slew rate. How do i know what power to use?
        //  displayTextLine(15,"%f",abs(*desiredSpeed - *currentSpeed));
            if (*currentSpeed < *desiredSpeed){
                *desiredPower+=powerIncrement;
                motorspeed(*desiredPower, currentPower);//increase desired power
            }
            else{//decrease speed
                *desiredPower-=powerIncrement;
                motorspeed(*desiredPower, currentPower);
            }
        }
        sleep(50);
    }
}



task motorControlTaskL()
{
    // while (true){
    //  motorspeed(desiredPowerL,&currentPowerL);
    // }
    angularVelControl(&desiredSpeedL, &currentSpeedL, leftMotor, &desiredPowerL, &currentPowerL);

}

task motorControlTaskR()
{
    // while (true){
    //  motorspeed(desiredPowerR,&currentPowerR);
    // }
    angularVelControl(&desiredSpeedR, &currentSpeedR, rightMotor, &desiredPowerR, &currentPowerR);


}


void go(){
    setMotorSpeed(leftMotor, currentPowerL);
    setMotorSpeed(rightMotor, currentPowerR);
}
```

```
task gotask()
{
    while (true){
        go();
    }
}

task display()
{
    while (true){

        displayTextLine(0,"RMotor: %f/%f",currentSpeedR, desiredSpeedR);
        displayTextLine(8,"LMotor: %f/%f",currentSpeedL, desiredSpeedL);
    }
}


task main(){
    tolerance = .05;
    powerIncrement = 1;
    increment = 1;

    desiredSpeedL = 10;
    desiredSpeedR = 10;

    desiredPowerL=5;
    desiredPowerR=5;

    startTask(display, 10);
    startTask(gotask, 10);
    startTask(motorControlTaskR, 10);
    startTask(motorControlTaskL, 10);
    while(true){
    }
    // daniel test

}
```

2. This question most of the code is actually related to the screen display instead of the actual angle change. Since setMotorSyncEncoder does all of the heave work for you (since you can just pass it the angle and it'll turn the desired amount) the only movement task is go, which just calls anglechange. The updateAngle task just constantly gets the angle gone from the right motor encoder. getAngularVelRad is essentially convertRad from the previous program but we get the sign manually since we were told that there was an error with the recommended way. The display task just calculates the various screen elements and displays them. When we tested with the recommended angles we found that this was pretty accurate, it might be off by a degree or so because of the sudden stop but it was different every time, sometimes it would stop dead on, other times it would be just slightly off. For a fairly crude method it was more accurate than I would have guessed.

```
#pragma config(StandardModel, "EV3_REMBOT");

float currentAngle;
float desiredAngle;
float desiredRadian;
float leftAngularVel;
float rightAngularVel;
float Xr;
float A;
float diameter;
float l;



float getAngularVelRad(tMotor motor){
    float first;
    float second;
    first = getMotorEncoder(motor);
    sleep(10);
    second = getMotorEncoder(motor);
    float sign =  first-second;

    if (sign<0){
        return -((getMotorRPM(motor)*(2*PI))/60);
    }


    return ((getMotorRPM(motor)*(2*PI))/60);
}

void anglechange(){
    if(desiredAngle<0){
        setMotorSyncEncoder(leftMotor, rightMotor, 100, (desiredAngle*2), 80);

    }
    else{
        setMotorSyncEncoder(leftMotor, rightMotor, -100, (desiredAngle*2), 80);
    }
    sleep(1000);
}

```

```
task display(){
    while (true){
        leftAngularVel = getAngularVelRad(leftMotor);
        rightAngularVel =getAngularVelRad(rightMotor);


        Xr = (diameter/2)*(rightAngularVel+leftAngularVel);
        A = (diameter/(2*l))*(rightAngularVel-leftAngularVel);


        displayTextLine(0, "leftVel: %f", leftAngularVel);
        displayTextLine(2, "rightVel: %f", rightAngularVel);
        displayTextLine(5,"Desired Angle: %d",desiredAngle);
        displayTextLine(7,"Current Angle: %d",currentAngle);
        displayTextLine(9,"the error %d",desiredAngle-currentAngle);
        displayTextLine(10, "A: %f", A);
        displayTextLine(12, "Xr: %f", Xr);

    }
}
task go(){
    anglechange();
}

task updateangle(){
    while(true){
        currentAngle= getMotorEncoder(rightMotor)/2;
    }
}



task main(){

    diameter = 58;
    l = 60;

    currentAngle=0;
    //desiredRadian=-((2*PI)/3);
    desiredRadian=2*PI;
    desiredAngle= desiredRadian*180/PI;
    startTask(display, 10);
    startTask(go, 10);
    startTask(updateangle, 10);
    while(true){}
}
```

3. The goal of this program is to essentially turn with a slew rate. It accelerates until it is at full power just like the normal slew rate. But it begins decelerating once k*error is less than the desiredPower. Once the error is within the tolerance, the entire program shuts down. We have a display task that keeps track of the various elements required on it, and we also have a separate task that keeps track of the angle and the error so that other tasks don't need to. As for K, we found that k's between 0 and 1 were the best, and that .5 was about right for us. At .5 it decelerates at a fairly moderate rate, whereas higher k's would decelerate much too fast, and lower k's would be much too slow. We also added a minimum power for deceleration. This was added because it would often get down to 1 power and still need to turn and it would just take forever, so 7 is still slow enough but it doesn't take forever to reach the target.

```
#pragma config(StandardModel, "EV3_REMBOT");

float currentAngle;
float desiredAngle;
float desiredRadian;
float leftAngularVel;
float rightAngularVel;
float Xr;
float A;
float diameter;
float l;
float k;
float error;
float tolerance;
float angleRemaining;//this is a percentage .
float angleGone;//percentage of angle gone
int currentPower;
int desiredPower;
float turnRate;
bool decel=false;
float decelpower;


float getAngularVelRad(tMotor motor){
    float first;
    float second;
    first = getMotorEncoder(motor);
    sleep(10);
    second = getMotorEncoder(motor);
    float sign =  first-second;

    if (sign<0){
        return -((getMotorRPM(motor)*(2*PI))/
    }


    return ((getMotorRPM(motor)*(2*PI))/60);
}
```

```
void decelerate(){
    int ratio;

    ratio = -100;

    if(desiredAngle<0){
        ratio = 100;
    }

    //float decelpower;
    while (true){
        decel=true;

        if(error<= tolerance){
            error=0;//this should stop the robot once error is with
            stopAllTasks();
        }



        if( (k* error) < currentPower){
            decelpower=k*error;
        }
        else{
            decelpower=currentPower;
        }
        if( decelpower<7){
            decelpower=7;
        }
        setMotorSync (leftMotor, rightMotor, ratio, decelpower);
        sleep(50);
        //We probably don't want it to sleep at all, but it may not
    }

}


void accelerate(){
    // accelerate until full power or there is only a percentage of
    int ratio;

    ratio = -100;

    if (desiredAngle<0){
        ratio = 100;
    }

    //&&  (angleGone<angleRemaining)
    while((currentPower<desiredPower) ){
        currentPower+=turnRate;
        setMotorSync (leftMotor, rightMotor, ratio, currentPower);
        sleep(100);
    }

}
```

```
task display(){
    while (true){
        leftAngularVel = getAngularVelRad(leftMotor);
        rightAngularVel =getAngularVelRad(rightMotor);


        Xr = (diameter/2)*(rightAngularVel+leftAngularVel);
        A = (diameter/(2*l))*(rightAngularVel-leftAngularVel);

        displayTextLine(5,"Desired Angle: %d",desiredAngle);
        displayTextLine(7,"Current Angle: %d",currentAngle);
        displayTextLine(9,"the error %d",error);
        displayTextLine(10, "A: %f", A);
        displayTextLine(12, "Xr: %f", Xr);

    }
}
task go(){

    accelerate();

    // maintain(); this may be unnecessary due to the k*error

    decelerate();

}

task updateangle(){
    while (true){

        currentAngle= getMotorEncoder(rightMotor)/2;
        angleGone = currentAngle/desiredAngle;
        error = abs(desiredAngle-currentAngle);

        // if(error<= tolerance){
        //  error=0;
        //  stopTask (updateangle);//maybe this should be stop
        // }
    }
}



task main(){

    diameter = 58;
    l = 60;
    k=.5;
    //tolerance=.1*180/PI;
    tolerance=2;
    angleRemaining = 1 - .25;
    turnRate = 3;
    currentPower=0;
    desiredPower=50;
```

```
    currentAngle=0;
    //desiredRadian=-((2*PI)/3);
    desiredRadian=2*PI;
    desiredAngle= (abs(desiredRadian*180/PI));
    startTask(display, 10);
    startTask(go, 10);
    startTask(updateangle, 10);
    while(true){}
}
```

4. This program was developed to move the robot forward or backward while using a slew rate. This helps preserve the gears and the motor by slowly increasing and decreasing the speed. We programmed our robot to analyze the speed, distance the robot has gone, and distance left to go. The robot accelerates all the time, until it reaches 100, The way we chose to do this was by taking some sample data by testing how long (distance) it took the robot to slow down at a certain speed. We used the speeds 100, 75, 50, and 25. We then looked at the best distance it took to slow the robot down using a slew of 3. Once we got all our distances recorded, we analyzed the data to develop a formula: .136*speed+2.4*speed*(3/slewrate). This tells the robot if you are going “X” speed, at this distance you need to slow down by the slew. This keeps going until the robot comes to a stop. Most of the pictures are on the next page

```
void distancetravled(){
    float leftgonedeg;
    float rightgonedeg;
    int reset=0;
    while (true){
        leftgonedeg=getMotorEncoder(leftMotor);
        rightgonedeg=getMotorEncoder(rightMotor);
        if(reset>=200){
            reset=0;
            totalleft+=(leftgonedeg*0.51);
            totalright+=(rightgonedeg*0.51);
            resetMotorEncoder(leftMotor);
            resetMotorEncoder(rightMotor);
        }
        reset+=1;
    }
}

task displayTask(){
    while (true){
        display();
    }
}
task test(){
    go(304.8);
}
task slewTask(){
    slew();
}
task trav(){
    distancetravled();
}
```

```
#pragma config(StandardModel, "EV3_REMBOT");
float speed;
float slewrate;
float totalleft=0;
float totalright=0;
float distance;
float disttomax=((((.136*speed+2.4)*speed)*(slewrate/3))*2);

void decrease(){
    if (speed-slewrate>=15){
        speed-=slewrate;
        }else{
        speed=10;
    }
    sleep(250);
}

void increase(){
    if (speed+slewrate<=100){
        speed+=slewrate;
        }else{
        speed=100;
    }
    sleep(250);
}
```

```
void slew(){
    while(true){
        disttomax=((((.136*speed)+2.4)*speed)*2)*(slewrate/3);
        if(totalleft>= distance/2){
            if(distance-totalleft<=disttomax || distance-totalright<=disttomax){
                decrease();
            }
        }else{
            increase();
        }
    }
}

void go(float dist){
    distance=dist;
    while (true){
                                        istance){

 task main(){
     speed=0;
     slewrate=3;                        >=distance){
     startTask (displayTask,10);
     startTask (test,10);
     startTask (trav,10);
     startTask (slewTask,10);
     while(true){}
 }                                      t);
                                        ght);
    displayBigTextLine(12,"STOP: %f", ((((.136*speed+2.4)*speed)*(slewrate/3))));
}
```

5. Truthfully the program we made for this question is an exact splice of question 3 with question 4 with only a few very minor changes. The programs work the exact same way as in questions 3 and 4 respectively, except that instead of passing numbers in directly to the programs, we pass in a variable which we change that is either the angle or the distance in millimeters. This allows us to reuse the program multiple times in this file. We also had to add reset functions for each program (to reset the variables that are used as bounds/conditionals). The program is much too long to include it's entirety in screenshots and it is nearly verbatim of questions 3 and 4 so if you need to refer to the pictures for each of those questions for parts omitted. I've added screenshots of the important parts that have been modified. PICTURES NEXT PAGE

```
void reset(){
    speed=0;
    slewrate=3;
    totalleft=0;
    totalright=0;
    stops=true;
}
```

```
void foward(float dist){
    reqdist=dist;
    startTask (trav,7);
    startTask (test,7);
    startTask (slewTask,7);
    while (true){
        if (!stops){
            stopTask(test);
            stopTask(trav);
            stopTask(slewTask);
            reset();
            return;
        }
    }
}
```

```
void reset2(){
    currentAngle=0;
    desiredAngle=0;
    desiredRadian=0;
    angleRemaining=0;
    angleGone=0;
    currentPower=0;
    decel=false;
    decelpower=0;
    stopturn=false;
    resetMotorEncoder(leftMotor);
    resetMotorEncoder(rightMotor);

}
```

```
void turn(float deg){
    desiredAngle= deg;
    theta+= deg;
    startTask(goes, 7);
    startTask(updateangle, 7);
    while (true){
        if (stopturn){
            stopTask(goes);
            stopTask(updateangle);
            sleep(500);
            reset2();
            return;
        }
    }
}



task main(){
    diameter = 58;
    l = 60;
    k=.5;
    tolerance=2;
    angleRemaining = 1 - .25;
    turnRate = 3;
    currentPower=0;
    desiredPower=5;
    currentAngle=0;
    slewrate=3;
    startTask(display,7);



    turn(60);
    foward(500);
    turn(-90);
    foward(1000);
    turn(150);
    foward(300);\


    stopAllTasks();











    while(true){}
}
```