To: Programming Group 5914

From: Larry Basegio

Subject: Code Discoveries of Tuesday 17 December

Date: 19 December 2019

Summary

Over several weeks during the 2019 post season the concept of using execution threads (parallel execution) has been investigated with limited success. At times finding out what doesn't work can provide valuable clue(s) as to path. I believe this is the case here. Note that the purpose is to facilitate autonomous operation as well as automated positioning of certain functions while manually driving. This investigation began during the 2019 build season but was not implemented in competition.

Historical

Using the 2019 Robot the following methods were investigated:

- Creating separate execution threads for the operation of the drive, lift tilt, and inner lift. We had limited success with the tilt mechanism and the drive mechanism. Problems with the Redline encoder on the inner lift prevented investigation of this feature. The problems with the Redline encoder have since been resolved.
- Creating a sensor reading thread and using these values to operate the other three functions (drive, tilt, lift). This thread cycled through the sensors (gyro and three encoders) and updated variables in real time. The goal was to be able to use these values as needed during program execution. The problem encountered was that there was no way to stop the thread other than shutting down the computer. This is a significant problem. After consulting with Chief Delphi, it was confirmed that this is a bad idea. A better approach would be to use a new feature of the robot application, e.g., the function robotPeriodic() included within the source file Robot.java.
- This approach was investigated last Tuesday. Using robotPeriodic() produced a similar problem. It would continue to read the sensor values until the robot was powered down. If the program was halted within the editor, the last value read would be implemented the next time the program was started, i.e., the sensor would not reset. This indicates that robotPeriodic() may be another thread similar to that listed in bullet two. There is not an obvious way of insuring that it terminates prior to termination of main().

Threads

Execution threads have the following properties.

- They operate in parallel with the main() thread time slicing between thread of equal priority.
- Delays implemented within a given thread do not delay other threads operating in parallel other than the delays associated with the time sharing property.
- "Child" threads, the threads we are speaking of, must terminate prior to the termination of the main() thread. Java language provides significant support for creating and using threads.

Tuesday 19 December 2019 Insights

We attempted to use the function robotPeriodic() as a real-time “sensor reading” function. Every tenth (or hundredth) time this function was called we would output the sensor values. This worked as advertised but we encountered the issues listed earlier in “Historical”. We attempted to use these values to drive the robot forward a fixed distance in autonomous mode. Specifically, the function within the class listed in robotDrive.java, i.e., moveFwd() was asked to move the robot forward 5 feet. Note that this function contains a while() loop that compares a real time sensor value with a calculated sensor value to tell it when to stop moving. The short version is that the robot did not stop and did not travel straight indicating that the control loops were not functioning.

- While locked in the while() loop the sensor values did not update – once instructed to move forward it continued to move forward having never reached the target encoder value. This is a manifestation of sequential operation as compared to parallel operation.
- While locked in the while() loop, the gyro heading did not update, therefore heading correction was not functional.
- robotPeriodic() did not stop when the program terminated – the last value read from the sensors was maintained on the next startup. This probably has something to do with the robot application itself. One had to completely exit VS to remedy this situation.

Thoughts

I return to one of the basic properties of the “child” thread, that is, a delay within a child thread does not affect execution timing of other threads operating in parallel (other than the timing slicing referred to earlier). For the example attempted last Tuesday, the while loop within the function driveFwd() would not have hung the rest of the program if implemented as a finite-duration thread. Java provides methods for insuring that child threads terminate prior to the main() thread. A simple example is provided below. In this example of a simple thread values of x are incremented and the value of y is calculated (a simple quadratic). There is a while() loop that allows the calculation to continue as x is incremented as long as the value of y remains below 100. I will add that the main() of this application created and destroyed two different calculation threads (this quadratic and another cubic thread) several times. This illustrates that multiple while() loops can be run in parallel, one not affecting the other. I can demonstrate this program to anyone interested. The central point here is that we should proceed with creating such child threads for individual operations. These threads can be operated sequentially or in parallel for both autonomous and teleoperated application.

```java
////////////////////////////////////////////////////////////////////
//  File:  FirstThread.java
/////////////////////////////////////////////////////////////////////
//
//  Purpose:  Illustrates the creation of a thread that runs
//            outside of main().  Note that delays within the
//            thread do not delay the main thread.  Also note
//            that the main thread must complete after any of
//            the child threads (like this one) complete or
//            a nasty system crash will result.
//
//  Remarks:  1.  We should be able to use the join() function
//                to wait for the child thread to complete.
//                This compiler doesn't like it.  It must be
//                contained within a try/catch block.
//            2.  Adding a small delay in the run() portion allows
//                other threads to intersperse with this one.
//            3.  In terms of using this concept to control the
//                robot, it might allow the lifts and tilts to
//                perform their movements while the operator can
//                drive the robot to the position.
//            4.  Watchdogs will need to be disabled??  Not sure.
//                When we are in a thread outside the robot
//                application we will be manipulating motors until
//                they acheive the position.  In many ways it is like
//                a while() loop but the main() program doesn't hang.
//                As previously mentioned the thread must complete
//                prior to the exit of the main() thread.
//
//    03/13/2019:  Changed for() loops to while() loops.  Added
//    escape counters - a break statement when the count is exceeded
//    will take us out of the while() loop.
//
//
/////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////
```

```java
import java.util.*;

class Compute1Thread implements Runnable  {
    String name;
    Thread t;
    Runtime r=Runtime.getRuntime();
    private Delay delay;


    //  Constructor
    Compute1Thread(String threadname)  {
        name=threadname;
        t=new Thread(this,name);
        System.out.println("New thread: " + t);
        delay=new Delay();
        t.start();  //  Start the thread
    }

    //  Entry point for the thread.  This is where any desired
    //  action needs to be implemented.  It will run in "parallel"
    //  or "time share" with other active threads.
    public void run()  {
        double x;
        double y=0;
        int i=0;
        int count=0;

        while(y<100.0)  {
            x=MultiThreadTest.x_start + (i*.1);
            y=4*x*x - 7*x + 5.0;
            System.out.println(name + " i = " + i + " x = " + x + " y = " + y);
            //  This small delay allows the second thread to intersperse with
            //  this one.  Without the delay, this thread might run to completion
            //  before the other thread gets to work.
            delay.delay_milliseconds(1.0);
            i++;
            count++;
            if(count>100)break;
        }
        System.out.println(name + "Exiting");

        //  Initiate garbage collection.
        r.gc();
    }
}
```