| Design | Advantages | Disadvantages |
|---|---|---|
| Design 1 | • Easy to use either polar or cartesian coordinates | • Methods contain many if statements since either type of coordinates could be stored |
| Design 2 | • Quick retrieval of polar coordinates | • Cartesian coordinates take time to calculate and must be calculated each time<br>• getDistance and rotatePoint methods run slower |
| Design 3 | • Quick retrieval of cartesian coordinates<br>• getDistance and rotatePoint methods run quicker | • Polar coordinates take time to calculate and must be calculated each time |
| Design 5 | • Code is easy to modify and scale (ie. adding other coordinate systems, adding common methods) | • More boilerplate code that may not be necessary for functionality |

| Methods | Design 1 | | | Design 2 | | | Design 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Min | Med | Max | Min | Med | Max | Min | Med | Max |
| getX() | 0 | 0.067 | 0.073 | 0.064 | 0.066 | 0.070 | 0 | 0 | 0.009 |
| getY() | 0 | 0.069 | 0.071 | 0.057 | 0.069 | 0.072 | 0 | 0 | 0.003 |
| getRho() | 0 | 0 | 0.002 | 0 | 0 | 0.003 | 0 | 0 | 0.009 |
| getTheta() | 0.003 | 0.005 | 0.380 | 0 | 0 | 0.002 | 0.374 | 0.582 | 0.588 |
| getDistance() | 0 | 0.271 | 0.275 | 0.263 | 0.271 | 0.280 | 0 | 0 | 0.003 |
| rotatePoint() | 0.541 | 0.689 | 0.708 | 1.263 | 1.286 | 1.311 | 0.539 | 0.550 | 0.562 |

| Methods | Design 5 Sub2 | | | Design 5 Sub 3 | | |
|---|---|---|---|---|---|---|
| | Min | Med | Max | Min | Med | Max |
| getX() | 0.065 | 0.067 | 0.072 | 0 | 0 | 0.004 |
| getY() | 0.068 | 0.070 | 0.078 | 0 | 0 | 0.010 |
| getRho() | 0 | 0 | 0.003 | 0 | 0 | 0.003 |
| getTheta() | 0 | 0 | 0.002 | 0.375 | 0.582 | 0.592 |
| getDistance() | 0.266 | 0.270 | 0.277 | 0 | 0 | 0.003 |
| rotatePoint() | 1.264 | 1.287 | 1.307 | 0.539 | 0.551 | 0.558 |

The tests were conducted 10 times with each test consisting of every design calling every method 10,000,000 and measuring the nanosecond runtime of each method. The larger number of calls was necessary to ensure that the runtime would be long enough to convert to seconds.

The results of the test were mostly as expected. Design 2 and Design 5 Sub 2 had negligible runtimes for getRho and getTheta. While Design 1 and Design 5 Sub 3 had negligible runtimes for getX and getY. This is because those methods required no calculations and simply returned the specific attribute of the method.

The getX and getY methods for designs that had to calculate them were still relatively fast since those methods did not require large calculations. On the other hand, the getTheta method was the longest getter, most likely because of the complex equation involved. What surprised me was the runtime for getRho which despite involving a complex equation, had negligible runtime.

For similar reasons as above, the getDistance and rotatePoint methods were the fastest when the design stored cartesian coordinates since those methods only use the getX and getY methods. However, overall the rotatePoint method is very slow, most likely due to the complex equation involved.

Since the runtimes for getRho() were not what I expected, I debugged the code to double-check that the runtime calculations were correct. I noticed a spike in runtime whenever I put a breakpoint anywhere in a test method even if it was after the runtime calculations. I concluded this was most likely due to it using more system resources. In the end, all the code and runtime calculations seemed to be working correctly and the log files generated after each test support this conclusion. As such, I believe that the Math.sqrt and Math.pow functions used in calculating rho are significantly faster than the Math.toDegrees and Math.atan2 used to calculate theta.