Brewing Quality in Android Robots

Nikhil Murthy[1], Anshuman Radhakrishnan[1], Simon Chow[1], Siddharth Suri[1], Sameer Suri[1] and Kingsum Chow[2]

batteriesinblack@gmail.com[1], kingsum.chow@intel.com[2]

Abstract

The emergence of versatile embedded processors and the Internet of Things bring new opportunities to the robotic world. The organization For Inspiration and Recognition of Science and Technology (*FIRST*) [1] that runs worldwide robotics competitions for K-12 students has recently announced changes in the programming platform for the *FIRST* Tech Challenge (FTC) robot game. They now adopt Java and Android running on commonly embedded devices that are similar to smartphones. The new programming platform not only makes the robot platform better, it also allows us to study the challenges of testing software quality in the modern robotic world. Using the case study of the FTC robot game, this paper describes the taxonomy of robot testing processes for software quality, covering several key areas: (1) testing robot software and sensor quality in the presence of uncertainties, (2) a systematic approach to identify defects in the performance of Android robots and (3) developing and testing machine learning programs for autonomous robots that improve decision making.

Biography

Batteries in Black is a robotics team formed in 2007. After four years in FIRST LEGO League (FLL), a LEGO robotics competition for elementary and middle schools, the team moved to FIRST Tech Challenge (FTC). The team has been recognized many times for being an exemplary team in the robot competition, community outreach, and software development. In the past two years, the team was nominated for the Control Award, an award that recognizes the programming and software achievements of the team, at the World Championships. In 2015, the team received the Sensor Savants Judges Award at the World Championships, recognizing the extensive research and effort put into developing and programming custom sensors.

Kingsum Chow is a Principal Engineer at Intel. He works in the System Technology and Optimization division of the Software and Services Group. He joined Intel in 1996 after receiving a Ph.D. in Computer Science and Engineering from the University of Washington. Since then, he has been working on performance, modeling and analysis of software applications. In his spare time, he volunteers to coach multiple robotics teams to bring the joy of learning Science, Technology, Engineering and Mathematics to the students in his community.

[1] FTC Team 4855 Batteries in Black, https://www.facebook.com/FTCBatteriesInBlack

[2] System Technologies and Optimization, Software and Services Group, Intel Corporation

1 Introduction

There are many programs and organizations around the world that aim to promote learning science, technology, engineering and math (STEM). Martínez et al [2] conducted an experiment with Preschool and Elementary students to determine how one's age affects the ability to learn basic robot programming. Additionally, the success of learning science through robotics is explored by a couple of publications by Niu et al [3] and Chow et al [4]. Phelim Dowling and Kevin McGrath [5] also describe an approach using free and open source tools to manage the efficiency and quality for planning and creating software projects. Another major organization that promotes STEM around the world is FIRST. FIRST stands for "For Inspiration and Recognition of Science and Technology" and FIRST is a worldwide robotics program that is aimed towards kids ranging from kindergarten to 12th grade. By using their knowledge gained from FIRST, the authors describe how a team should divide into specialized sub teams in order to increase the overall efficiency. The use of freely available resources and integrating those tools to maintain software quality is crucial as the popularity of learning science and engineering through robotics increases.

The First Tech Challenge (FTC) is one of the two options for high school or experienced middle school students who want to participate in the FIRST program.

The FTC robot game is composed of two phases: (1) the autonomous phase and (2) the user control phase. In the autonomous phase, the robot independently makes decisions about movement and scoring actions (e.g. picking up balls, or putting balls in goals). The challenges of testing robots in this phase include testing combinations of objects in the environment to simulate a great number of scenarios, and monitoring the robot's motions and decisions. In addition, sensor errors and variations of robot motions make testing the quality of software quite difficult. In the subsequent user control phase, team members use a pair of game controls and a tablet running a user-written software application to control the robot. In this phase, the main challenge is to be able to react and adapt accordingly based on the opponents' movements in order to outscore them.

1.1 Introduction to Robot Programming

Robots utilize many different types of computer controllers, making use of devices that have been specifically designed for the application of robotics. In FTC, the robot's motions are controlled by Android smartphones and tablets, which are programmed in Java. The Android device sends commands to different types of motors in order to make the robot move. To aid with decision making, the phone also can receive input from internal and external sensors.

The Android device on the robot is programmed from a computer, using Android Studio and the FTC SDK, which adds functionality to control motors from the phone. Both autonomous and user control programs are loaded onto the robot, and then are run from the Android device. During the autonomous phase, the robot moves based solely on its autonomous program. During the user control phase, the Android device on the robot is connected to another Android device through Wi-Fi Direct. This second Android device is connected to two joysticks and transfers joystick input to the Android device on the robot. From there, the drivers of the robot can use their joysticks to drive the robot. Teams use the Android software platform on the mobile phones located both on the robot and with the driver to run their programs that control the robot. A diagram illustrating the interaction between the different components of this system is below:

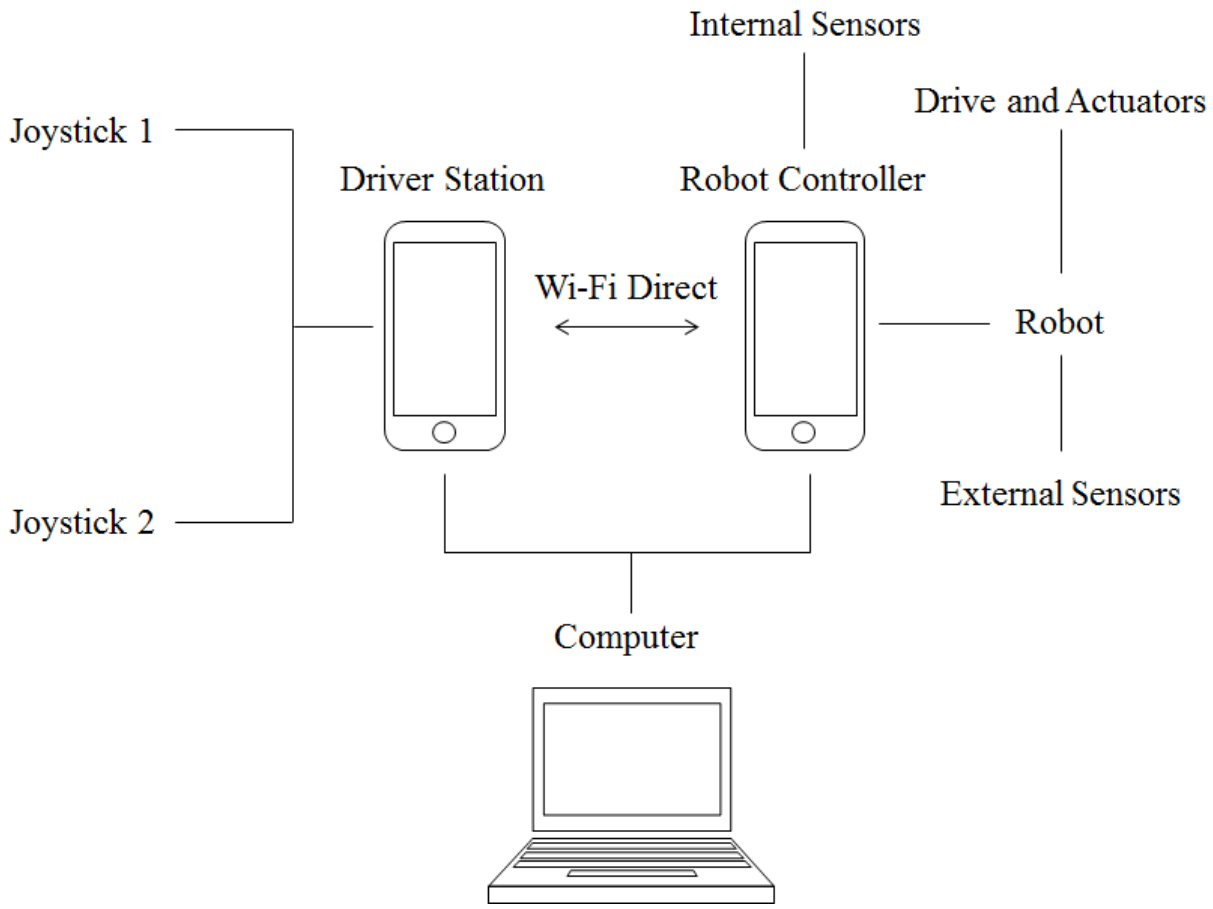

Figure 1. The control system of an FTC robot and its subsystems consisting of joysticks (used by drivers), Android smartphones with internal sensors, a robot with external sensors and actuators, and a computer for programming.

1.2 The Robot Components

The robot can be broken down into several components such as the hardware, software, internal sensors, and external sensors, and the communication among them. The hardware consists of the physical mechanisms that allow the robot to interact with the playing field and objects on the field. For example, actuators and drive trains are both part of the hardware. The software consists of the instructions given to the robot, whether the robot is operating autonomously or under human influence. These instructions can be altered by the use of sensors and these instructions can identify potential decisions. Internal sensors are built into the Android phone on the robot and thus are consistent between all robots on the field. These sensors include magnetometers and accelerometers. External sensors are attached to the robot but not to the phone directly. The communication components present sensor data from multiple sources to the robot controller. Android phones contain a multitude of built in sensors [6]. Some are used to collect motion data, others measure conditions in the environment, and some are used to measure the physical location of the device. For example, the Android phone used in FTC has a three-axis accelerometer. The light sensor and proximity sensor collect environmental data.

Each of these sensors collects different types of data which can be used by the robot to know its location in the environment, as well as states of the environment. The three-axis accelerometer records the current acceleration along the x-, y-, and z-axes of the phone. This is useful to be able to find the motion of the phone, based on changes of acceleration over time. The gravity sensor returns a vector that gives the direction of gravity, which shows the orientation of the phone in respect to the earth. The three-axis

magnetometer records the magnetic field of the earth, also on its x-, y-, and z-axes. This can also be used to find the motion and orientation of the phone. Using motion and orientation data from these sensors, it is possible to constantly calculate the position of the phone as it moves.

The light sensor measures ambient light in front of the phone, which can be used to understand the conditions of the phone's environment. The proximity sensor is a low-resolution distance sensor which returns whether or not an object is within 5 cm of the phone. This sensor gives a simple indication if your phone is close to an object. These two sensors allow the phone to get some simple information about the environment, which can help the robot know how to interact with its environment. In addition to sensors built into the phone, external sensors such as ultrasonic sensors, inertial measurement units, and infrared seekers can be useful.

2 Challenges of Brewing Software Quality in Robots

Testing and developing software for robots is inherently difficult due to the difficulty in isolating the precise point of failure, whether it is present in the hardware, software, or electrical systems. Robots also constantly evolve in terms of hardware and physical capabilities, thus software must be reliable and adaptable. In addition, field conditions vary, so the robot has to be able to adapt to a variety of conditions and make reliable decisions. This presents two main challenges in developing software for robotic competitions.

The first challenge is ensuring that internal sensors are reliable. Internal sensors, as defined above, are sensors built into the Android phones that are in use for the robotics competition, such as accelerometers and magnetometers. It is key that these sensors are reliable and can be characterized so that failure can be identified quickly for any of the components. To ensure that the variance between sensors on different phones is minimal, different phones need to be tested.

The second challenge is in ensuring that the robot can accurately make decisions in a variety of situations. This challenge is difficult to solve, because the robot potentially needs to accumulate knowledge from previous scenarios, and use the data from the current scenario to make an informed decision. Since so many factors change between each run of the robot, decision-making has to be tested thoroughly to ensure that the robot can consistently make the right decision.

3 A Taxonomy of Robot Testing Processes

Internal sensor reliability is a key challenge for the robot testing process. Internal sensors consist of the built in sensors in Android phones in the case study laid out in this paper, but these concepts can be applied to a general case in which sensors are used. By looking at the inherent error built into the sensors from stationary testing, the reproducibility of sensor readings through multiple trials, and the variance across multiple devices and sensors by performing similar tests on multiple Android phones, testing of the sensors is fully characterized. Robot function reliability consists of the robot's performance itself, its reproducibility, and the interaction between the software and physical elements. The robot decision reliability is critical in order to perform autonomously and minimize error, and consists of machine learning algorithms and decision-making techniques.

By using several case studies including the development of several robots in the context of the FIRST Tech Challenge (FTC) competition, the robot testing process and taxonomy is revealed. A black box approach through data logging is important in order to effectively and efficiently characterize the stimulus (input) and response (output) of the behavior. Testing within each category should cover three main classifications: the Physical World, the Software World, and the Interaction World (the interface between the other two worlds). The use of real and simulated hardware allows for extensive testing both before and after hardware has been developed.

3.1 Testing the Reproducibility of Sensor Readings on Mobile Phones

A case study involving ZTE Speed Android phones and their built in sensors will further illustrate the taxonomy of the robot testing processes. These phones contain numerous sensors including an accelerometer, linear accelerometer, and magnetometer. Phones were evaluated on a test bed that consisted of a rotating platform that spun the phone in a consistent manner while the phone recorded sensor readings. We also recorded data of a stationary phone as our control, to compare to the data from the spinning phone, and to see inherent error in the sensor.

Figure 2. The test bed used to collect sensor data while the phone is rotated. A LEGO Mindstorms EV3 system is used to drive the rotation.

Figure 2 illustrates the testing rig connected to an LEGO Mindstorms EV3 system [6]. To test the reproducibility of movement with the case study of the ZTE speed phones, the consistency of data across multiple trials and different phones was tested. The test bed above was constructed to minimize error during data collection. For example, the turns that move the phone are powered by a LEGO motor, therefore allowing for controlled motion by another robot in comparison to movement by hand which brings in human error. While collecting data from the many sensors, the phone was rotated to different sized turns to fully test sensor readings ranging from stationary (0 degrees) to one rotation (360 degrees). These tests were repeated many times to analyze the consistency of the readings, and were repeated on another Android phone. Each phone was turned four times at each 90, 180, 270, and 360 degrees while collecting data. Both of these phones had the same sensors including an accelerometer and a magnetometer, allowing the devices to be directly compared. The data collected on each was compared to look at the reproducibility on different devices, a variable important in robotics.

Stationary tests in which the phone did not move were conducted. Here, the deviations from the mean were calculated for each sensor, allowing for an approximation of the error due both to environmental and non-environmental causes (i.e. sensor noise) that would be present in other tests. Table 1 illustrates the stationary tests (across multiple trials and phones) and each sensor's corresponding standard deviation:

Table 1. The standard deviations of internal Android sensor data of stationary tests. Tests were conducted on two phones and across 12 sensors.

Sensor	Standard Deviation Phone 1	Standard Deviation Phone 2
Accelerometer x-axis	0.22370 m/s^2	0.02261 m/s^2
Accelerometer y-axis	0.07897 m/s^2	0.02655 m/s^2
Accelerometer z-axis	0.16040 m/s^2	0.06752 m/s^2
Gravity x-axis	0.02313 m/s^2	0.00978 m/s^2
Gravity y-axis	0.02003 m/s^2	0.00568 m/s^2
Gravity z-axis	0.00057 m/s^2	0.00010 m/s^2
Linear Accelerometer x-axis	0.08300 m/s^2	0.02487 m/s^2
Linear Accelerometer y-axis	0.04811 m/s^2	0.02273 m/s^2
Linear Accelerometer z-axis	0.12271 m/s^2	0.04518 m/s^2
Magnetometer x-axis	0.23700 µT	0.34770 µT
Magnetometer y-axis	0.20070 µT	0.17742 µT
Magnetometer z-axis	0.24363 µT	0.28269 µT

In addition to stationary tests, repeated tests of different sized turns were done. These turns are 90, 180, 270, and 360 degrees and are done by a NXT robot with a high level of accuracy. This eliminated error that could arise if the phone was turned by hand. The sensor values were analyzed and compared to trigonometric functions due to the circular nature of the turn. The following graph, for example, shows the x-axis accelerometer data for a full 360 degree turn and the corresponding trend line:

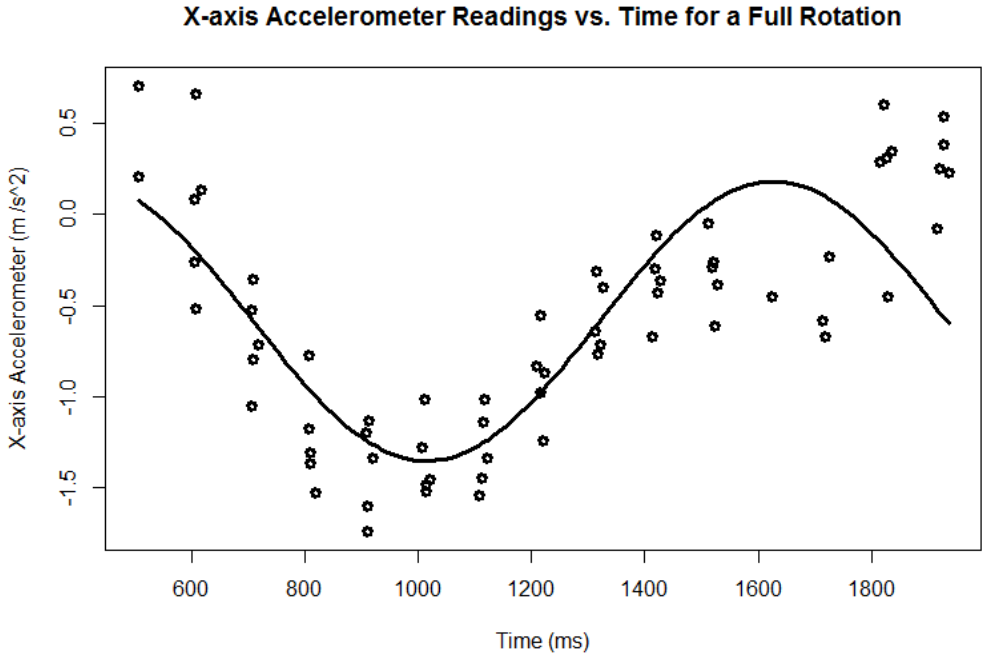

Figure 3. Sensor data of the x-axis accelerometer versus time over a full rotation on the first Android phone. The scatterplot highlights the trigonometric trend, which is further backed up by a regression: y=-0.58645-0.76761sin (0.4998-0.005136x) with a coefficient of determination of 0.9465.

The trend line is modeled by the equation: y=-0.58645-0.76761 sin (0.4998-0.005136x) and shows high correlation. The coefficient of determination for this model is 0.9465, signifying its high trigonometric correlation. Similarly, the same was done for the following sensors that showed a trend: Accelerometer (three axis) and magnetometer (three axis). The other sensors, such as the gravity z-axis sensor, had a constant value around 9.81 m/s^2, as expected. In addition to modeling these sensors' data and looking at their correlation, different phones and their reproducibility were analyzed. The graph below highlights the x-axis accelerometer readings on a second phone and a trigonometric model of the data:

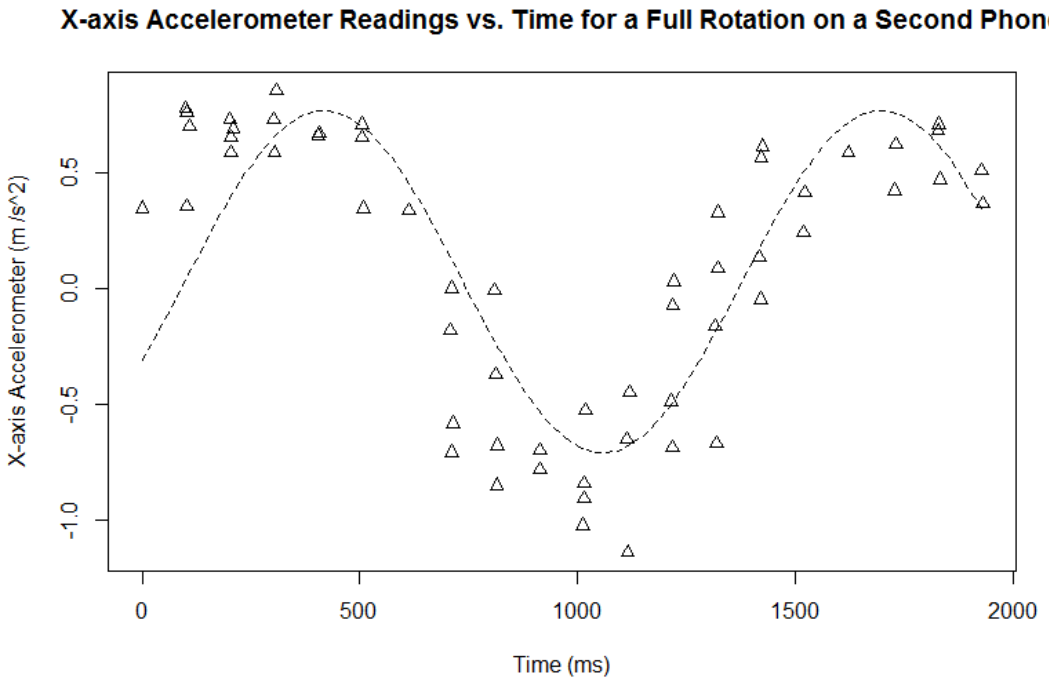

Figure 4. Sensor data of the x-axis accelerometer versus time over a full rotation on the second Android phone. The graph features a trigonometric trend, which is also shown by the trend line: y=-0.58645-0.76761sin (0.4998-0.005136x) with a coefficient of determination of 0.9465.

The trend line above is modeled by the equation: y = 0.02785-0.73899 sin(0.4800-0.00491x) and has a coefficient of determination 0.805135, highlighting its high correlation. The error in both phones from the theoretical sine wave to the data can be explained by multiple factors. First, the average standard deviation of both phones of x-axis accelerometer stationary data is 0.123155 m/s^2, which highlights the error of the sensors either deriving from environmental or inherent causes. Secondly, some error can be explained by the irregularities of the turn by the robot. In theory, the sine wave approximates a turn in which no acceleration occurs (by the NXT motor). This is physically impossible, and is indicated on the graphs by data points towards both the beginning and end that are farther away from the trend curve.

The phone's trend curves can be directly compared to one another by looking at their coefficients and graphs. The following table describes the coefficients of the general sine wave equation:

For the equation, y = a + b sin(c + dx):

Table 2. Table of coefficients of each phone's trend line in the general form of: y = a + bsin(c + dx). The percent difference between the two is also listed, allowing the two curves to be directly compared.

Coefficients:	a	b	c	D
Phone 1	-0.58645	-0.76761	0.4998	-0.005136
Phone 2	0.02785	-0.73899	0.4800	-0.004910
Percent Difference	2206%	3.87%	4.13%	4.60%

As shown in Table 2 the main difference between the two models is the coefficient a as all other coefficients have a percent difference of less than 5% (versus over 2000%). The two graphs also highlight this difference, seen by the apparent upward shift derived from the change in the coefficient, a:

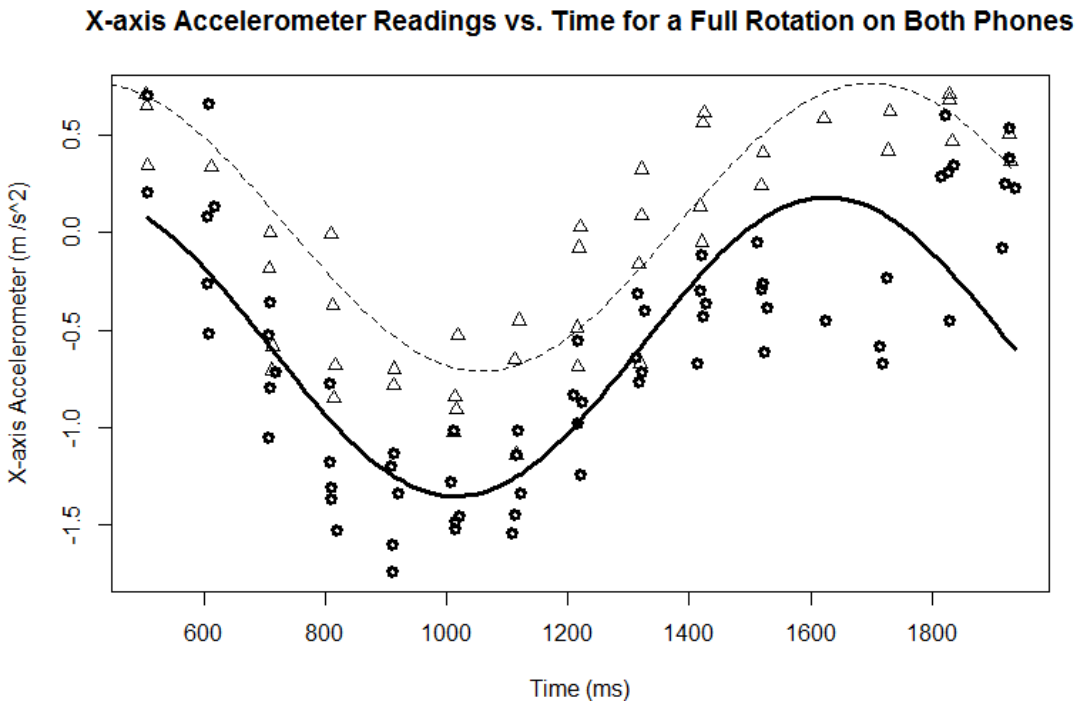

Figure 5. Sensor data of the x-axis accelerometer versus time over a full rotation on both Android phones. The first Android phone data is represented by the solid line and circles while the second Android phone data is represented by the dashed lines and triangles.

From looking at the variation in sensor data for circular motion, it was found that this data is very reliable and consistent across phones and across multiple trials. Although this is true, values are initially offset and therefore should be calibrated by accounting for the offset, which is roughly equal to the coefficient a in the general sine wave equation listed above.

Unlike testing with circular motion, testing for arbitrary motion was not very accurate. For testing arbitrary motion, we performed several experiments that involved two phones held together with rubber bands. The goal of banding them together was to negate the possibility of one phone moving away from the other. By negating the change in position between the two phones, we inferred that any discrepancies between the two data plots would have come from the uncertainty within the phone's data collection systems.

After finishing the data collection on our phones, we realized that another source of error for the experiment was that the two phones did not begin data collection at the exact same time, thus resulting in slightly inaccurate data. To fix this problem, we included a timestamp in our data analysis. This changed the independent variable of our data from time elapsed since data collection began, to time elapsed since January 1, 1970. This timestamp allowed the data from the two phones to be lined up with each other as they have the same starting point and this minimized the uncertainty and error in our data analysis.

After implementing the timestamp, we proceeded to graphing our processed data. Each phone had a separate graph for the x-axis, y-axis and z-axis accelerometer values. Since we were experimenting with two phones, this resulted in six individual graphs. It was observed that there was little difference between the data of the two phones.

In order to clearly show the error between the two data sets, we took the difference between each of the corresponding data values that were collected by the two phones. We also performed the same set of actions for the magnetometer x, y and z values. This resulted in six additional graphs as there are two sensors with each having three components to it. We analyzed the spread of the data for each of the six data tables by finding their standard deviation. Once we saw the standard deviations greater than three for the accelerometer data, we were surprised, as that means that our accelerometer data is very spread out and inaccurate. However, after researching a bit more about the phone sensors and the inaccuracy of the accelerometer we decided to analyze the gravitational sensor as well. The gravitational sensor seemed to be a more processed and therefore accurate version of the accelerometer and once we created the "difference" graphs for the gravitational sensor, we could clearly see the gravitational differences were much smaller than the accelerometer differences as there were more points that had zero difference on the graph with the gravity sensor.

Table 3. Standard Deviation Table of the Differences between the Two Phones for the Three Sensors and Their X, Y and Z Axis Values

Sensor	Mean	Standard Deviation
Accelerometer - X (m/s^2)	0.1489	3.560
Accelerometer - Y (m/s^2)	-0.1677	3.290
Accelerometer - Z (m/s^2)	0.2718	3.193
Gravity Sensor - X (m/s^2)	0.3771	0.8091
Gravity Sensor - Y (m/s^2)	-0.1863	0.7193
Gravity Sensor - Z (m/s^2)	0.5159	0.7990
Magnetometer - X (uT)	0.7935	13.70
Magnetometer - Y (uT)	-0.7022	19.36
Magnetometer - Z (uT)	0.5822	13.67

As illustrated in Table 3, the standard deviations for the magnetometers are comparatively high while the gravity sensors' standard deviations are extremely low and accurate. From our data tables and graphs, we conclude that the phone is accurate, but not accurate enough for robot and software testing. In

robotics, one only needs a few data points to be way off in order to declare the sensors not reliable enough. One way we can increase the accuracy in the future is to disregard the outliers. For example, when running robot or a software and collecting data, we could ignore data values that are over two standard deviations away, or over one standard deviation away, if we want it to be more accurate. However, we will have less data points if we implement a change like this.

3.2 Testing the Decision Making Process of a Robot

The increase in the number of sensors available to the robot and the complexity of the changes in the robot's environment create opportunities to use machine learning to make decisions. Machine learning [1][9] has been shown to perform the job well for human activity recognition. We extend the work to robotics. In FTC, sensor usage scenarios were subject to variation in motions on the field and environment changes. We specifically use machine learning to classify the position of a field element into a certain position out of a given number of possible positions. In using infrared seekers, we found that they can have readings offset by bright lights and variations in the battery level of the target object associated with an infrared beacon. We use a classification machine learning algorithm to smooth out the effects from these sources of variation.

We used one of the machine learning methods called the logistic regression [6]. Our approach, as illustrated in Figure, consists of two phases, the offline training phase and the online scoring phase. With logistic regression, we found it was particularly easy to implement the scoring phase.

Figure 6. Process for creating Logistic Regression Algorithm

In FTC, a robot may be in an unstable position during the autonomous phases if it bumps into other objects on the field. If the robot is in an unstable position, we want to return it to a flat position and regain balance. This involves first classifying the robot's current orientation, and then implementing a correction procedure. Using built-in sensors like the accelerometer and magnetometer, data samples were collected for the robot tilted in various directions: no tilt, tilted forward, backward, left, and right. Sensor data was developed into features and logistic regression was used to classify the orientation of the robot.

We implemented a logistic regression with regularization **Error! Reference source not found.**. Essentially, a sigmoid function is fitted to the data by updating the parameters so that a cost, or error, function is minimized. This predicts the probability that the input is in a certain class. A common problem with Machine Learning Algorithms is that they either suffer from high bias, where the prediction function underfits the data (an example of this would be when a linear function acts as the prediction function when the data is roughly quadratic), or they suffer from high variance, where the prediction function overfits the data (an example for this would be if the prediction function is a fifth-order polynomial, and the data is roughly quadratic). In essence, underfitting models the data too poorly, and is thus unable to generalize to new inputs, and overfitting fits the data too well, and is also unable to generalize to new inputs. These problems can be solved in different ways-- bias can be solved by increasing the number of features, and variance can be solved by reducing the number of features, or collecting more data. In order to avoid overfitting the data, as a preventive measure, we used a method called regularization.

Regularization helps solve the overfitting problem by increasing the value of the cost function when the parameters have large values. This forces the parameters to shrink, and thus reduce complexity- in essence, reducing the number of features. Thus, it minimizes the effects of overfitting and results in an optimal algorithm.

As we plan to use the same algorithm in multiple phones, we conducted the experiments to quantify sensor variations within a phone and across phones. We mimicked the changing of the orientation of the robot by changing the orientation of the phones. Five phones were used to collect data to quantify sensor variation across phones in addition to sensor variation within a phone. Each phone was identical in model, and each phone was tested in the same way. The robot is essentially a rectangular prism. Figure illustrates one of the five orientations of the phone mimicking the robot. We designate five classes corresponding to the normal and four different off-balance positions. The five classes are: Class 1 - Flat (Normal, No Tilt), Class 2 - Tilted Head Up, Class 3 - Tilted Head Down, Class 4 - Tilted Right Up, and Class 5 - Tilted Left Up.

Each phone was placed into these five positions. Data was collected in each of the positions at 10 times a second, leading to over 50 samples for each data set (a data set corresponds to data from one phone, for one orientation) for a large number of built-in sensors on the phone. After data collection, we had 25 data sets. A picture of our testing apparatus for orientation 3 is shown in Figure.

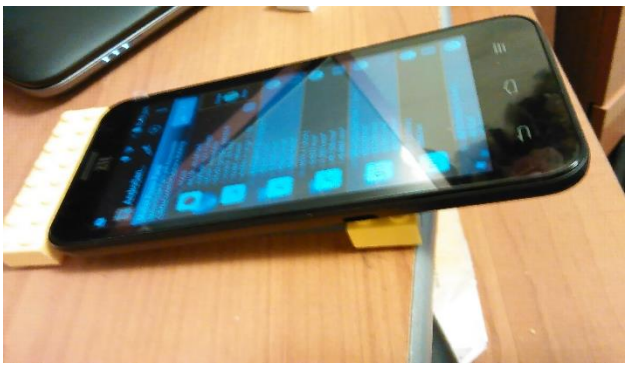

Figure 7. An example of data collection for machine learning. Orientation position 3 is shown.

Before analyzing the algorithm, several features were eliminated, as data from certain sensors was identical between different orientations. This was a rudimentary feature selection, though more complex methods [11] of eliminating and weighting features are available.

Another element of data processing before the machine learning algorithm was the simplification of the remaining features. We recognized that since the phone was stationary for each data set, the values never strayed considerably from their initial value. Thus, we took the initial value from twelve sensors, leading to twelve features. Our reduced data set for one phone looked like this:

Table 4. Reduced data from one phone: initial value for each attribute.

Accelerometer (m/s^2)			Linear Accelerometer (m/s^2)			Magnetometer (uT)			Orientation (°)			Position
X	Y	Z	X	Y	Z	X	Y	Z	Yaw	Pitch	Roll	
0.01	-0.08	9.45	0.00	0.01	-0.35	-36.96	1030.4	-123.48	2.06	0.40	0.02	1
-0.10	1.13	9.39	-0.01	-0.07	-0.38	-43.08	1027.5	-120.96	2.44	-7.00	0.50	2
-0.08	-1.32	9.39	0.02	0.02	-0.36	-26.16	1035.7	-126.66	1.55	8.17	0.58	3
2.05	-0.10	9.25	-0.05	0.05	-0.33	-25.56	1036.7	-123.90	359.95	0.67	-12.27	4
-2.42	-0.09	9.22	0.03	0.03	-0.32	-10.80	1038.6	-124.92	2.33	0.43	14.66	5

The data was then used to train the machine learning algorithm. Data from three phones formed the basis for training, and the remaining 10 datasets were reserved for testing. The selection of the phones was arbitrary. We wanted to ensure that the algorithm would be accurate enough for any phone's data, so fifteen data sets were used for training. The remaining 10 data sets were tested to see if the algorithm

could correctly classify the orientation of the robot. For example, data collected when another phone (a phone that was not used in training) was placed in Orientation 4 is shown below.

Table 5. Data from one Phone, in one Orientation. The values shown represent the first value of each feature, the same as the training data.

Accelerometer (m/s^2)			Linear Accelerometer (m/s^2)			Magnetometer (uT)			Orientation (°)		
X	Y	Z	X	Y	Z	X	Y	Z	Yaw	Pitch	Roll
2.242	0.185	9.474	-0.021	0.004	-0.066	9.853	-12.367	-53.473	242.440	-1.016	-13.320

The algorithm finds the function that outputs the highest probability for the data and predicts that the robot is in that class. Here is a table that shows the actual orientation of the phone, and the predicted orientation. As shown below, the algorithm was 100% accurate.

Table 6. This is a "Confusion Matrix", which shows the actual orientation of the phone, and then the predicted orientation. Values outside the diagonal represent errors, but as can be seen below, since all values are in the diagonal, the algorithm was 100% accurate for the ten testing examples.

		Predicted Class				
		Orientation 1	Orientation 2	Orientation 3	Orientation 4	Orientation 5
Actual Class	Orientation 1	2	0	0	0	0
	Orientation 2	0	2	0	0	0
	Orientation 3	0	0	2	0	0
	Orientation 4	0	0	0	2	0
	Orientation 5	0	0	0	0	2

After testing the algorithm on data from other phones, a quick review of the prediction functions and the original data sets revealed that the data sets were similar for a large number of sensors, and there was no appreciable difference in the readings of the sensors except for a select few sensors. Thus, the majority of features were not very useful. In future applications of this algorithm, the number of features will probably be reduced to minimize overfitting, especially since only a small amount of features is necessary for accurate classification.

The main challenge with this application of machine learning was the similarity of the data sets, which led to the outputs of the prediction functions being very similar between classes. For example, when a data set for position 4 was tested, the algorithm gave a high probability of the robot being in either position 3 or 4, as these positions differ largely only in the readings for a few sensors. Ultimately, the probability of being in position 4 was higher, so the algorithm could still correctly classify the test data. As mentioned above, further iterations will either eliminate some features or develop a way to weight the importance of features.

Further action would involve more advanced feature selection in our machine learning algorithms, along with the implementation of different statistical methods to experiment with eliminating overfitting. Other applications of machine learning include the analysis of the tilt of the robot to determine its exact orientation in space-- in other words, to determine its pitch and roll (which would both be zero if the robot were flat). This would act as an extension of the above application of machine learning. Another machine learning application would be in image processing, an area that we have researched before, but which we have yet to apply machine learning to.

In conclusion, the decision making of the robot can be improved by using machine learning algorithms, which allows the robot to learn from past decisions and make informed and accurate decisions in a

variety of situations. This decision-making process is important to test in order to ensure high performance when the robot runs autonomously.

4 Summary

Our approach to brewing quality in Android robots that integrates robot testing processes. Case studies were used to examine quality in Android devices in the context of the FIRST Tech Challenge robot competitions and develop a testing taxonomy. Internal sensors within Android phones including the accelerometer and magnetometer and their data were analyzed and collected under controlled conditions such as precise turns. With each movement, a theoretical model of the expected sensor output was generated based off of the characteristics of motion. For circular motion, this model was a trigonometric sine wave for both the magnetometer and the accelerometer. These motions were carried out by a robot in order to ensure reliability. It was found that by comparing a regression of the data (in the form of the mathematical model) and the data, the precision of the sensor data could be quantified. In addition to controlled motion, arbitrary motion was conducted in order to test reliability of sensor readings across phones (and sensors). By applying these concepts to the reliability of a robot, the consistency of performance was determined by utilizing the established testing taxonomy. Finally, the decision making of the robot that allows it to function autonomously was refined by employing machine learning algorithms. Statistical methods were employed in order to ensure that prediction algorithms could accurately make decisions based on prior data. By testing in many different scenarios, we ensured that our robot wouldl be able to use an accumulation of data from past scenarios to adapt to new and unexpected situations. All of these methods can be extended to industries that require the usage of sensors with minimal adjustments.

The contributions of this paper are:

- Established a taxonomy of robot testing processes
- Developed an approach to integrate robot testing processes with regard to inherent sensor, robot, and robot decision reliability.
- Demonstrated a prototype in FIRST Tech Challenge robot competition

Acknowledgments

The authors wish to express their gratitude to their reviewers, Aaron Hockley and Moss Drake, whose assistance and feedback have been invaluable.

References

[1] http://www.usfirst.org/
[2] M. Cecilia Martínez, Marcos J. Gómez and Luciana Benotti. A Comparison of Preschool and Elementary School Children Learning Computer Science Concepts through a Multilanguage Robot
[3] Vicki Niu, Ethan Takla, Ida Chow, MacLean Freed, Jeffery Wang and Kingsum Chow. Engineering Quality in the Robotic World. Proceedings of the Pacific Northwest Software Quality Conference, 2012. Portland, Oregon, USA.
[4] Kingsum Chow, Ida Chow, Vicki Niu, Ethan Takla and Danny Brillhart. Software Quality Assurance in the Physical World. Proceedings of the Pacific Northwest Software Quality Conference. 2010. Portland, Oregon. USA.
[5] Phelim Dowling and Kevin McGrath. Using Free and Open Source Tools to Manage Software Quality - An agile process implementation. ACM Queue April 2015.
[6] http://developer.Android.com/guide/topics/sensors/index.html
[7] http://www.lego.com/en-us/mindstorms/about-ev3
[8] Pierluigi Casale, Oriol Pujol, and Petia Radeva. “Human Activity Recognition from Accelerometer Data Using a Wearable Device”. Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, 2011, pp 289-296 http://dx.doi.org/10.1007/978-3-642-21257-4_36

[9] Mannini, A.; Sabatini, A.M. Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers. *Sensors* **2010**, *10*, 1154-1175.
[10] http://www.holehouse.org/mlclass/06_Logistic_Regression.html
[11] http://homes.soic.indiana.edu/natarasr/Courses/AML/Readings/FeatureSelection.pdf
[12] http://qwone.com/~jason/writing/lr.pdf