On the Road to Reliability for Android Robots in a Physical World

Krishna Saxena[1], Torin Perkins[2], Catherine Lee[3], James Menezes[4] and Bo Li[5]

bytesofkitkats@gmail.com, bo.a.li@intel.com

Abstract

Robotics software reliability is very crucial to control robot hardware while interacting with the physical world to accomplish a set of missions. The software needs to tackle senor errors, incorrect functionalities, software exceptions and uncertainties from the external environment. This paper depicts the challenges of attaining software reliability in Android robotics through the experiences of FIRST Tech Challenge (FTC) robotics team, the Bytes of Kitkats.

The software fault tree analysis (FTA) and software failure modes and effects analysis (SFMEA) are accomplished to prepare the statistical testing to evaluate and improve the software reliability. Software reliability is employed to predict the software reliability growth model (SRGM) through the stochastic process. The objectives are to build the software reliability model with finite or fixed number of inherent defects and accelerate the software reliability improvement process to ensure the robot behaves consistently in the field. Through the case study of building and testing Android robots with different motors, sensors and 3D-printed attachments, authors categorized sensor reading uncertainties (e.g., the position of robot and surrounding objects, etc.) and multiple uncontrollable environmental factors from the physical world in software reliability modeling.

Biography

Authors are high school students from the Bytes of Kitkats robotics team which was formed in 2013. In 2016, the team received Oregon championship award in FIRST LEGO League (FLL), a LEGO robotics competition for elementary and middle schools. In 2016, the team moved to FTC. The team has been recognized many times for being an exemplary team in the robot software development and innovate designs in Oregon, US FTC West Regionals and FTC world Championship events. In addition to striving for hardware and software excellence, the Bytes of Kitkats are also focusing on outreach activities in educating young students in Portland communities, other US states and countries oversea about robotics through camps, specific topic workshops, community open houses, and mentoring younger robotics teams.

Bo Li works in software developments and managements for Logic Technology Development in Technology and Manufacturing Group (TMG) at Intel. He joined Intel in 1999 after receiving Ph.D. from Georgia Institute of Technology. Since then, he has been focusing on software development, reliability, and testing. Outside of work, he mentors robotics teams in the community for high school and middle

[1] Jesuit High School, Portland, Oregon
[2] Central Catholic High School, Portland, Oregon
[3] Sunset High School, Portland, Oregon
[4] Westview High School, Portland, Oregon
[5] Logic Technology Development, Technology and Manufacturing Group, Intel Corporation

school students to foster more passions and knowledge in Science, Technology, Engineering and Mathematics (STEM).

1 Introduction

It has long been recognized that experiential and hands-on education provides superior motivation for learning new material by providing real-world meaning to the otherwise abstract knowledge. Robotics has been shown to be a superb tool for hands-on learning, not only of robotics itself, but of general topics in Science, Technology, Engineering, and Mathematics (STEM). Learning with robotics gives students an opportunity to engage with real life problems that require STEM knowledge. Mataric et al [1] describe the approach to enabling hands-on experiential robotics for all ages through the introduction of a robot programming workbook and robot test-bed aiming at providing readily accessible materials to K-12 for direct immersion in hands-on robotics. Additionally, the success of learning science through robotics is also explored by Chow et al [2] based on their experiences.

FIRST is among the broad spectrum of avenues for pursuing robotics at the pre-university level to promote STEM around the world. FIRST stands for "For Inspiration and Recognition of Science and Technology" and FIRST is an international youth organization to develop ways to inspire students in engineering and technology fields [3]. The FIRST Tech Challenge (FTC) is designed for students in grades 7–12 to compete using a sports model. Teams are responsible for designing, building, and programming their robots to compete in an alliance format against other teams. The robot kit is programmed using Java. Teams, including coaches, mentors and volunteers, are required to develop strategy and build robots based on sound engineering principles. The ultimate goal of FTC is to reach more young people with a lower-cost, more accessible opportunity to discover the excitement and rewards of STEM.

1.1 Background for FIRST Tech Challenge Robot Programming

The FTC competition field is 12' x 12'. Each match is played with 4 randomly selected teams, 2 per alliance. 4 18" x 18" robots must be able to navigate around each other without breaking when hit by another robot.

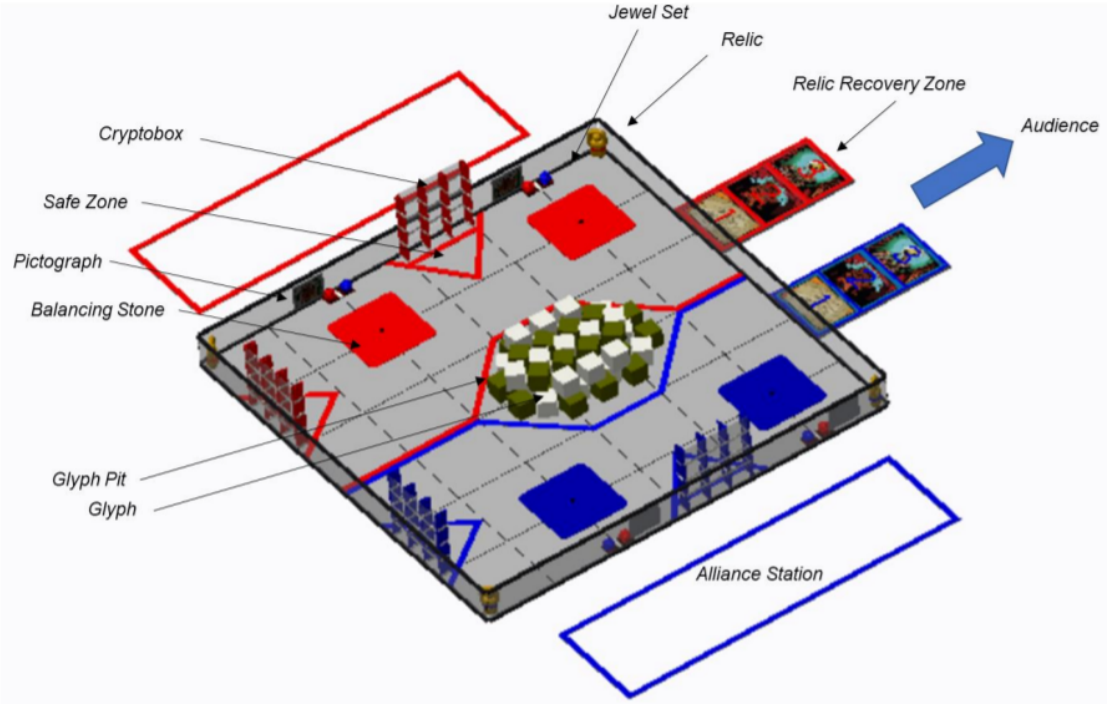


Figure 1. FTC field setup for the 2017-18 FTC season [4]

Figure 1 shows the field setup for the 2017-18 FTC season. The FTC robot game is composed of two phases: (1) the autonomous phase and (2) the user control tele operation phase. There are 2 red and 2 blue balancing stones being mounted on the field. Teams must balance their robot on their balancing stones prior to the start of the match. Teams also preload one glyph per robot. Field personnel then randomize the location of the jewels. They also randomly select one of three pictograph designs and install them on the field walls. During the autonomous period there are many ways for teams to score using only pre-programmed instructions in the phone mounted on the robot. If only one jewel is left on the platform at the end of the autonomous period, the alliance corresponding to that color will earn 30 points. Each glyph scored in a cryptobox earns the alliance 15 points. The pictographs have a coded message indicating which of the cryptobox column is a key. If a robot can decode this message and place a glyph in the cryptobox key, their alliance earns a 30 point bonus. Each robot parked in a safe zone earns their alliance 10 points.

In FTC, the robot motions are controlled by Android phones. The Android phone 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 software development kit (SDK), which adds functionality to control motors from the Android phone. Both autonomous and user control tele operation programs are loaded onto the robot, and then are run from the Android phone. During the autonomous phase, the robot moves based solely on its autonomous program. During the user control tele operation phase, the Android device on the robot is connected to another Android device through Wi-Fi connections. This second Android device is connected to two joysticks and transfers joystick input to the Android phone 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. During the tele operation period in this FTC season, the robot needs to collect glyphs and stack them up into the cryptobox to score points. A diagram illustrating the interaction among different components of this system is shown below (See Figure 2):

Figure 2. The control system of an FTC robot and its subsystems consisting of joysticks (used by drivers), Android smartphones with pre-loaded software programs, core components (e.g., external sensors, motors, power, etc.) to detect and control FTC robot [5]

1.2 Introduction to FTC Software Reliability

Software reliability is often defined as the probability of failure free operation of a computer program in a specified environment for a specified period of time [6]. FTC software reliability is very crucial due to the high expectations of consistent performance during FTC competitions for the new and updated versions of autonomous and tele operation control software. Software reliability is also a very time critical requirement, and the software failure can significantly impact FTC competition results.

Through the case study of building and testing a robot with motors and sensors in FTC, we characterized the robot sensor reliabilities and variable environmental factors in the physical world (e.g., the robot setup and locations, glyph pile setup, and balance stone conditions, etc.). Software reliability for FTC robots is difficult due to the difficulty in isolating the precise point of failure and evaluating the fault conditions for defect removal. In addition, robot design consistently evolves in terms of its hardware and physical capabilities during the competition season. Moreover, field conditions vary, so the robot has to be able to adapt to a variety of conditions with reliable software functionalities.

This paper presents two main challenges in accomplishing software reliability for FTC robotic competitions. The first challenge is ensuring that the software defects are removed effectively and the software can be adapted to the new capability and environment changes rapidly to reach the software reliability target. The second challenge is ensuring that the robot can utilize the software algorithms to minimize effects of sensor variability and analyze the physical world inputs to make right decisions in varieties of different situations. Since there are so many factor changes between each run of the robot, the software algorithms have to be tested thoroughly to ensure the robot can consistently make the right decisions and have very reliable performances.

2. On the Road to Software Reliability in Android Robots

In this paper, we used the fault tree as a logic diagram to display the interrelationships between a potential critical event (top event) in a system and causes for this event. The goal is to find critical events concerning fault introduction and take appropriate actions to reduce the probabilities of these events and the subsequent probability of the top event. A Fault Tree forms a logical representation of the manner in

which combinations of basic events, often made up of failures at the component level, could lead to a hypothesized failure of a system. We then think about these potential problems and how to detect, prevent, or mitigate the problems before they create a catastrophe which is to drive software failure mode and effects analysis (SFMEA). Reifer was widely credited for introducing the SFMEA to software engineering for requirements analysis in 1979 [7]. During SFMEA, possible design failure modes and sources of potential nonconformities must be determined in multiple software artifacts. A complete FMEA for a software-based system should, however, include both hardware FMEA and software FMEA, and the effects should be assessed on the final system functions.

Software FMEA in practice is often performed at different levels (i.e., system, subsystems, and components) which corresponds to architectural partitions or levels of abstraction. As design and implementation proceed, more details of the system are revealed, which enables more meaningful low-level analysis. Neufelder identified 8 viewpoints: functional, interface, detailed, maintenance, usability, serviceability, vulnerability, and software process [8]. The functional viewpoint is mostly useful for software FMEA, the interface viewpoint is for software/software and software/hardware interface FMEA, and the detailed viewpoint is applicable to design and implementation FMEA. The outcome of an FMEA is documented in a worksheet. Failure modes need to evaluate (a) severity which is subjective rating and measures how bad or serious the effect of the failure mode is, (b) occurrence which is the likelihood that the failure cause occurs in the course of the intended life, and (c) detection which is a subjective rating that quantifies the likelihood that a detection method will detect the failure of a potential failure mode before the impact of the effect is materialized. The combination of the severity, occurrence, and detection ratings is used to prioritize potential failure modes and root causes. The software FMEA team then agrees on a threshold of these factors and need to address all items above the threshold.

After the fault tree and software FMEA analysis, Software Reliability Growth Models (SRGMs) are considered to predict the defect density. Over the past decades, many researchers have discussed SRGM assumptions, applicability, and predictability. The basic assumption in software reliability modeling is that software failures are the result of a stochastic process, having an unknown probability distribution. Software reliability models specify some reasonable form for this distribution, and are fitted to data from a software project. Once a model demonstrates a good fit to the available data, it can be used to determine the current reliability of the software, and predict the reliability of the software at future times.

This paper employed the shortcut model prediction approach based on input answers to key questions. The Shortcut Defect Density Prediction Model [9] assumes that the defect density is a function of the number of risks versus strengths regarding this release of the software. The shortcut model includes factors that affect failure process. Using real-life data sets on software failures, this proposed approach is evaluated and compared to the real existing data from FTC competition matches. As a result, the applicability of the model is validated on these software failure data.

The software size is estimated in terms of 1000 source lines of code (Kilo-SLOC or KSLOC) first, then the associated defect density was selected based on the shortcut model. Here is the sequence of actions to get the predicted defect density. Table 4 shows Strength and Risk Survey for the shortcut model:

a. Count the number of yes answers in the Strengths. Assign 1 point for each yes answer. Assign 0.5
   point for each “somewhat” answer.
b. Count the number of yes answers in the Risks. Assign 1 point for each yes answer. Assign 0.5 point for each “somewhat” answer.
c. Subtracting the result of risk points from the result of Strength points.
d. If the result ≥ 4.0, predicted defect density = 0.110,
   If the result ≤ 0.5, predicted defect density = 0.647,
   Otherwise predicted defect density = 0.239 in terms of defects per normalized effective KSLOC.

The resulting defect density prediction is then multiplied by the normalized effective KSLOC (EKSLOC) to determine the defect density. EKSLOC is a weighted average of new, modified, reused, and auto-generated code. Here is the formula to normalize KSLOC for software reliability model effectiveness:

EKSLOC = New KSLOC + (A × major modified KSLOC) + (B × moderate modified KSLOC) + (C × Minor modified KSLOC) + (D × reused KSLOC without modification) + (E × auto-generated KSLOC)
where

- Reused code: Previously deployed on a previous or similar system without modification.
- New code: Code that has not been used in operation.
- Modified code: Reused code that is being modified.
- Auto generated code: Code that is generated by an automated tool.

The new code is 100% effective. Code that is reused without modification is fractionally effective as there may still be a few latent defects in that code. Code that is reused with modifications will have an effectiveness that is between these two extremes. Reused code that is subject to major modifications, for example, may be almost as effective as new code. Reused code with minor cosmetic changes, for example, may be slightly more effective than reused code with no modifications. Auto-generated code typically has an effectiveness that is similar to reused and not modified code since it is generated by a tool.

Table 1: Typical Effectiveness Multipliers

Multiplier	Ranges	Function of
A—major modification	≥ 40%	Magnitude of requirements change.
B—moderate modification	20% to 40%	Magnitude of design change and cohesiveness of design.
C—minor modification	5% to 20%	Cohesiveness of code (ability to change it without breaking an unrelated function).
D—reused	0% to 30%	Cohesiveness of design. How long the reused code has been in operation.
E—auto generated	0% to 10%	The ability to program the automated tool to provide code as per the design and requirements.

Normalized EKSLOC is then normalized for languages (Table 2) so that it can be multiplied by the predicted defect density that are in terms of defects/normalized EKSLOC to generate the predicted number of defects.

Table 2: Typical Ratios for Different Languages and Assemblers

Language type	Normalization conversion
High order language such as C, Fortran, etc.	3.0
Object oriented language such as Java, C++, C#	6.0
Hybrid	4.5

3. Case Study: FTC Android Robot Software Reliability Analysis and Improvements

By using several case studies including the development of robots in the context of FTC competitions, the robot software reliability fault trees, software FMEA and shortcut model were utilized to improve the robot software reliability in the competition field. The thorough fault event analysis and software functional FMEA analysis are crucial to effectively characterize the software reliability behavior to cover three main perspectives including the software World, the Physical World, and how they interact with each other.

3.1 FTC Robot Autonomous Glyphs Collections and Delivery to Cryptobox

We executed the case study for FTC Robot autonomous glyphs collection and delivery to cryptobox to illustrate the software reliability improvement processes.

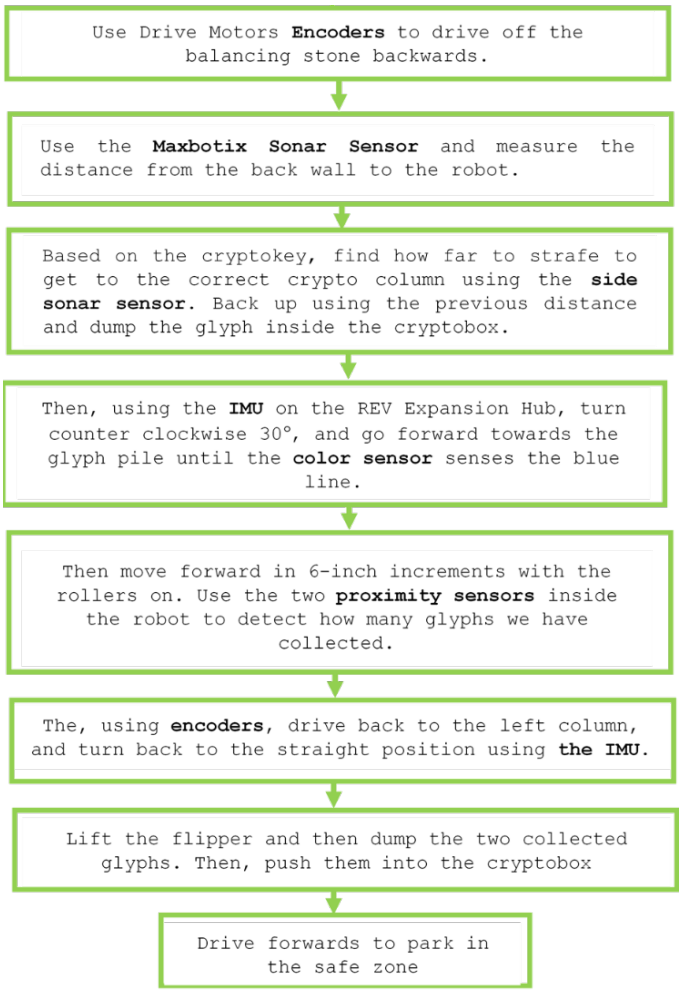
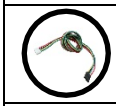


Figure 3. Robot Software Flowchart in Autonomous Period

Figure 3 shows the robot software flowchart for the software controlling FTC robot movements. The robot left the balance stone with one glyph within the robot. The target is to collect as many as possible (e.g., 2 or 3) of glyphs and deliver them to the cryptobox while the robot can't hold more than 2 glyphs at the same time. Figure 4 shows how the sequence of robot movements through the autonomous time period.

	Using **phone camera**, Vuforia SDK, OpenCV, identify the jewel, and flick it; Identify the crypto key.
	Use **encoders** on the drive motors to drive off the balancing stone.
	Use a Maxbotics **sonar sensor** to detect the robot's distance from the wall, and based on this distance, we strafed to the correct crypto column and dump the glyph.
	Using the Inertial measurement unit (**IMU**) on the expansion hub, we will turn about 45 degrees, making sure that we face the glyph pit.

	Use a **color sensor** to go forward until either the red or blue line. Then move forward while moving the rollers at the same time.
	Use two **proximity sensors** inside the robot to detect how many glyphs are collected from the glyph pile.
	Use **encoders** to go back to the cryptobox, the same way it came.
	Use an **IMU** to make the robot straight. Then lift the flipper lift and deposit the collected glyphs. Then move back into the cryptobox and score them.

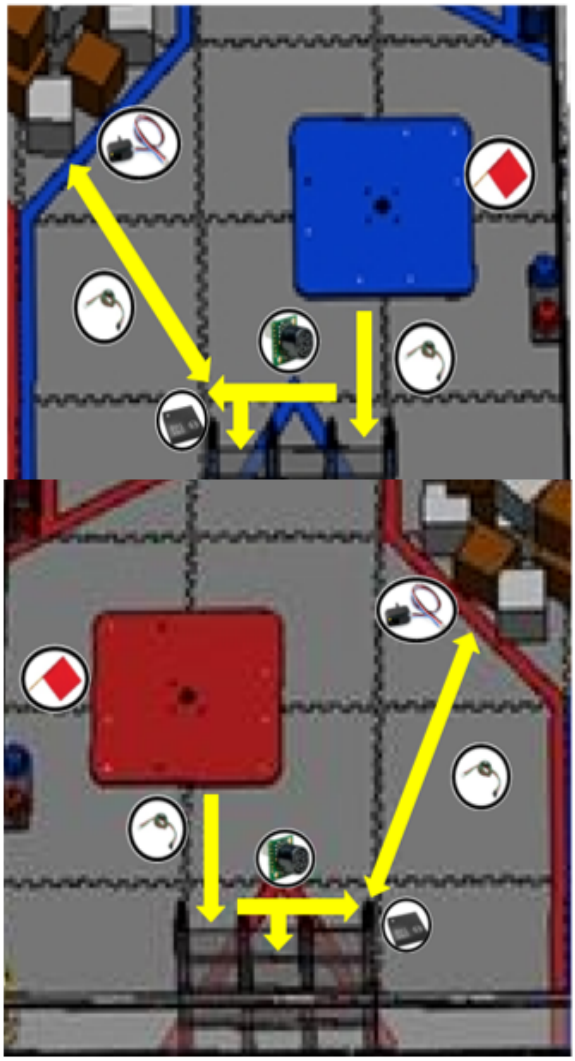


Figure 4. The Autonomous Glyphs Collections and Delivery to Cryptobox

The following graph (see Figure 5) shows how many glyphs the robot can deliver to the cryptobox throughout the 2017-2018 competition season for our robot. The team did a major software change before Houston World FTC championship competition in April'2018 in order to enable the robot ability to score more points during the autonomous time period. The shortcut software reliability model was employed to complete the defect predictions, and the team decided to complete the major software redesign with the understanding of the risk of small number of defects. Figure 5 illustrates that the software reliability is stable when the robot design and software algorithm intends to deliver only one glyphs to the cryptobox. The only incident happened during SC # 3 (Oregon State Championship Match #3) when the robot was positioned on the balancing stone incorrectly and later on was added to Fault Tree Analysis (Figure 6). Prior to Houston World Championship, the software was rewritten for ~ 50% of the code, the robot is capable of delivering 2 or 3 glyphs to the cryptobox. The testing spiral was rerun to cover scenarios based on fault tree analysis and software FMEA results for 4x2x3=24 test cases (Table 3). The differences among balancing stone, robot initial positions and cryptobox destination columns reflected the uncertainties in the physical world.

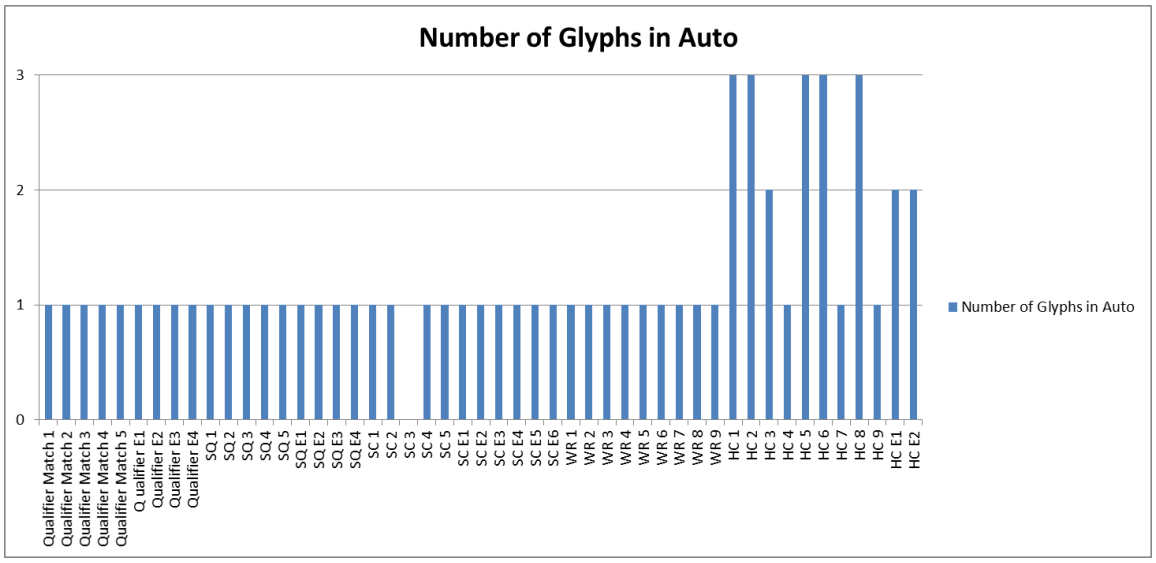


Figure 5. The number of glyphs that the robot can deliver in autonomous
(Qualifier Match: 1st level Oregon Qualifier Match
Qualifier: 2nd level Oregon Qualifier Match
SQ: Super Qualifier, 3rd level Oregon Qualifier Match
SC: State Championship, 4th level Oregon Qualifier Match
WR: US West Regional Match
HC: Houston World Championship Competition)
HC E1/2: Houston World Championship Semi-Final)

Table 3: Test Scenario Summary for Glyphs Collection and Delivery in Autonomous

Balancing Stone Selection (4 Balance Stones)	**Cryptobox Destination (either near alliance station or far away. See Figure 1)**	**Jewel Flick**
a. Red stone 1 (Left, Face Alliance Station)	a. Column 1 @ far Crytobox b. Column 2 @ far Crytobox c. Column 3 @ far Crytobox	a. Jewel on Left b. Jewel on Right
b. Red stone 1 (Right, Face Alliance Station)	d. Column 1 @ near Crytobox e. Column 2 @ near Crytobox f. Column 3 @ near Crytobox	
c. Blue stone 1 (Left, Face Alliance Station)	g. Column 1 @ far Crytobox h. Column 2 @ far Crytobox i. Column 3 @ far Crytobox	
d. Blue stone 2 (Right, Face Alliance Station)	j. Column 1 @ near Crytobox k. Column 2 @ near Crytobox l. Column 3 @ near Crytobox	

After these rigorous testing for different functionalities, the software reliability experienced one critical defect of the sensor failure with the wrong reading (HC #9) which caused the robot not able to collect more glyphs and then later on recover from this autonomous period fall out using the joystick remotely triggered software functions to recover the robot. Please note that HC #4 and #7 matches only delivered 1 glyphs because the glyphs pile is randomly setup and the robot intake mechanism can't collect 1 or 2 more glyphs through the pre-defined robot traveling path within 30 second autonomous. For HC #4 and #7, it is considered to be working successfully. Hence, the Mean Time between Failure (MTBF) is 720 seconds when considering both failures for SC#3 and HC #9. The # of defect matches the shortcut model prediction. The fault tree analysis (FTA) is shown in Figure 3. The software that we use to create the fault tree is from http://www.fault-tree-analysis-software.com/fault-tree-analysis [10]

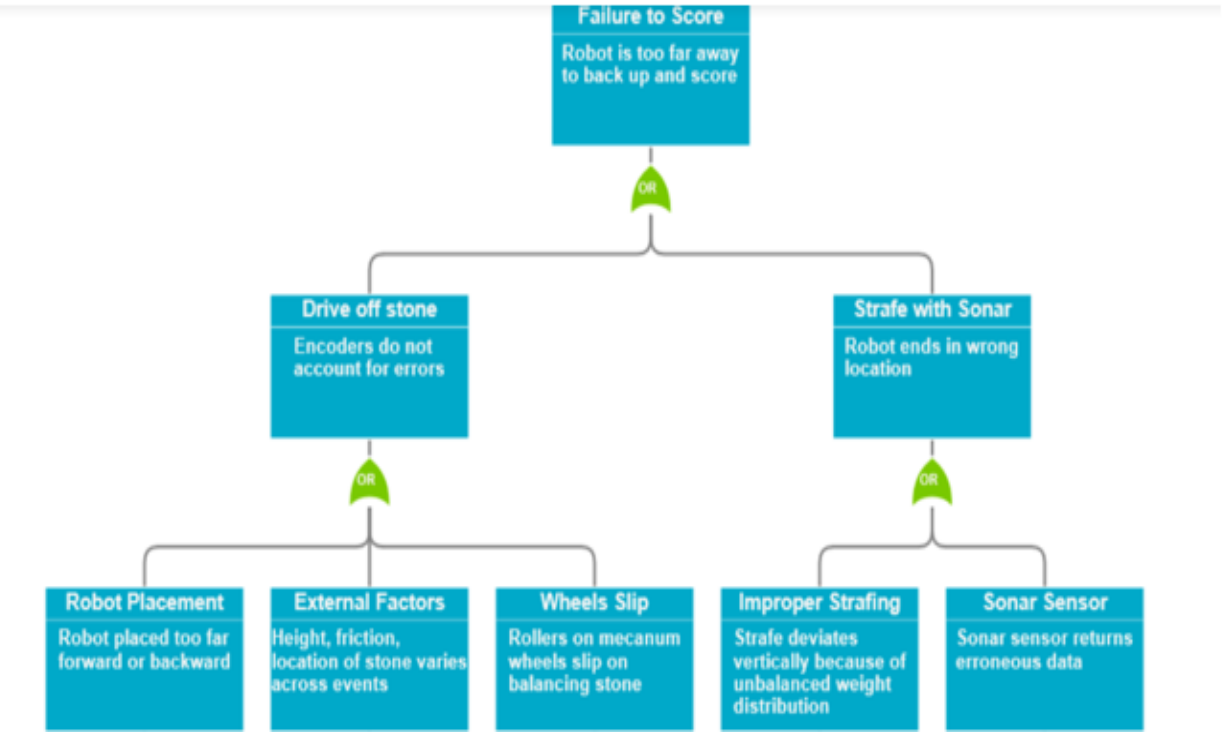


Figure 6. The Fault Tree to Collect and Deliver Glyphs to Crystobox in Autonomous

After we did fault tree analysis, we complete the Software FMEA for different viewpoints. Here is a sample software FMEA events for fault sequence and fault data category from the functional view point.

Table 4: Sample of Software FMEA Functionality Categorization and Action Implementations

Fault Category	Definitions	Key Fault Events Being Addressed
Functionality: Fault sequence/ order	A particular event is initiated in the incorrect order or not at all.	Fault Event: The robot lifting level in the software was set to zero at the incorrect lifting position due to the autonomous program stopped by the glyph Solution: Developed the software reset program and trigger the one button manual reset at the beginning of the tele operation (TeleOpe) session to recover in 1 sec.
Functionality: Fault data	Data is corrupted, incorrect, etc.	Fault Event: The robot back sonar sensor reading incorrect data caused the wrong traveling distance. Solution: added the error protection in the software program to cap within the threshold for the traveling distance to avoid hitting the balance stone.

The shortcut model was employed in the software reliability growth model and calculating the predicted defects. Here are answers to Table 5 shortcut model questions through the survey within Bytes of Kitkats team. The score delta between the software reliability “Strength” (9.5 points) and “Risks” (3.25 points) categories is 6.25 points > 4. Hence, the software defect density is expected to be 0.11 in terms of defects per normalized EKSLOC based on IEEE Recommended Practice on Software Reliability guidance. This information guided the team to do the following in order to reduce the risk so that the defect density can be reduced to lower value (0.11): a) The code reuse is considered in the robot hardware and software redesign for the world championship event. b) The team also ensures the software code is up to date and avoid using legacy code >10 years.

Table 5: The shortcut Model Software Reliability Answers to Survey Questions

Strength		**9.5 points**
1	We protect older code that shouldn't be modified.	Yes (1)
2	The total schedule time in years is less than one.	Yes (1)
3	The number of software people years for this release is less than seven.	Yes (1)
4	Domain knowledge required to develop this software application can be acquired via public domain in short period of time.	Yes (1)
5	This software application has imminent legal risks.	No (0)
6	Operators have been or will be trained on the software.	Yes (1)
7	The software team members who are working on the same software system are geographically collocated.	Yes (1)
8	Turnover rate of software engineers on this project is < 20% during course of project.	Yes (1)
9	This will be a maintenance release (no major feature addition).	No (0)
10	The software has been recently reconstructed (i.e., to update legacy design or code).	No (0)
11	We have a small organization (<8 people) or there are team sizes that do not exceed 8 people per team.	Yes (1)
12	We have a culture in which all software engineers' value testing their own code (as opposed to waiting for someone else to test it).	Somewhat (0.5)
13	We manage subcontractors—Outsource code that is not in our expertise, keep code that is our expertise in house.	No (0)
14	There have been at least four fielded releases prior to this one.	No (0)
15	The difference between the most and least educated end user is not more than one degree type (i.e., bachelors/masters, high school/associates).	Yes(1)
Risks		**3.25 point**
1	This is brand new release (version 1), or development language, or OS, or technology (add one for each risk).	Yes (1)
2	Target hardware/system is accessible within 0.75 points for minutes, 0.5 points for hours, 0.25 points for days, and 0 points for weeks or months.	0.75
3	Short term contractors (< 1 year) are used for developing line of business code	No (0)
4	Code is not reused when it should be	Somewhat (0.5)
5	We wait until all code is completed before starting the next level of testing	No (0)
6	Target hardware is brand new or evolving (will not be finished until software is finished)	Yes (1)
7	Age of oldest part of code >10 years	No (0)

Table 6: EKSLOC Normalizations for FTC Robot Software

	New KSLOC	**Major modified KSLOC**	**Moderate modified KSLOC**	**Minor modified KSLOC**	**Reused KSLOC**	**Auto generated code in KSLOC**	**Total EKSLOC**
Weight	100%	40%	30%	15%	5%	1%	
Software prior to April, 2018	5	0	0	0	0	0	5
Software post April, 2018 (HC)	2.5	0	0	0	2.5	0	2.5 + 0.125 = 2.625

Once the effective KSLOC is predicted, the last step is to normalize the EKSLOC based on the language used to develop the code (See Table 6). Some languages such as object oriented languages are denser than other languages. For FTC robot software, the Object oriented language Java was used. Hence, the normalization conversion factor is 6.0.The total EKSLOC is then shown as follows with the language type and normalization for FTC robot software reliability defect density predictions. The defect density prediction guided us in defining our robot redesign strategy and also the timing to redesign the software for the increased robot capabilities before the world championship competition to gain more scoring capabilities. The software reliability model helped us to evaluate the defect density with 1.73 defects (See Table 7). This guided the team to evaluate the risk of critical defects with the comparison against the original robot design and hardening process (3.3 defects in Table 7), and then make the informed risk-taking decision to redesign the robot hardware/software for the world championship event in Houston, TX. The team used the fault tree and FMEA to improve the robot software reliability in less than a month so that the robot can perform well in the field for the world championship competitions. Furthermore, this process helped the team to grow knowledge and skills to improve the software reliability and can be employed in other fields in the future.

Table 7: Final Computation for FTC Software Reliability EKSLOC Normalizations

Total EKSLOC	KLSOC	Normalization for this language	Normalized EKLSOC	Predicted defect density	Predicted defects = predicted defect density × normalized EKSLOC
Software prior to April, 2018	5	6	5	0.11	5 X 6 X 0.11 = 3.3
Software post April, 2018 (HC)	5	6	2.625	0.11	2.625 X 6 X 0.11 = 1.73

3.2 Autonomous Image Detection through Android Phone Camera Image and Reliable Software Algorithms

The accurate detections of VuMark and the two jewel locations are the keys to do the flicker action as part of the autonomous period. While a lot of FTC teams came up with designs to leverage different color sensors and complex software program, we utilized Android phone camera image and reliable software algorithms to find VuMark and the two jewel locations at a much faster speed of 0.25 second. As a result, this team eliminated unreliable sensor mount and reading errors, and also greatly simplified the software complexity to accomplish tasks reliably.

Figure 7. Self-correction Image Detection through Android Phone Camera Image (Left: ROI (Region of Interest) with BGR format; Right: Image of VuMark and the two jewels)

When we designed the solution to this complex problem, we found that the robot controller phones have cameras that can be accessed for computer vision purposes. As part of the design, we installed the phone mount on the robot frame. When we ran the autonomous code to initialize Vuforia software, the phone camera can view both the VuMark and the two jewels very reliably. Vuforia Software Development Kit (SDK) was then utilized to identify the VuMark to determine the correct crypto key. We then take the image in RGB format using Vuforia. Before inputting this image into OpenCV, we converted the image

from RGB to BGR, as OpenCV only processes in BGR format. Afterward, we converted the world coordinates to 2D image coordinates using Vuforia SDK. Once we know the coordinates of the center of the image, we identified the location of the Region of Interest (ROI) – the location of the left jewel in the image (See Figure 7). The rectangle containing the ROI is from (505,540) to (600, 640), as the center of the image was around (360,400) and the ROI covers almost 75% of the jewel. The ROI image is converted from BGR to Hue, Saturation, and Value (HSV) format. We then use OpenCV's histogram analysis API to give the histogram of the color of pixels in the ROI. We count the number of red pixels in the histogram, if there are more than half red pixels, we declare that the left jewel is red, otherwise it is blue. This case study is to show how to use the phone's camera to attain software reliability with very minimum hardware dependencies.

After we designed the phone camera based solution, we also completed fault tree analysis and took actions to address key fault areas to improve the software reliability and enable the consistent FTC robot performance in the field. In this case study, the ambient light fault event was addressed using the innovative software and hardware integrated solution. After creating a sample OpMode that plotted a histogram of BGR values using OpenCV, we saw that the histogram changed significantly when lighting conditions were changed which can be used to address ambient light fault event from Figure 8. After color representations, we chose to use HSV for color detection because the hue will be the same independent of the lighting conditions. The Opmode in FTC SDK is able to convert colors from the typical Red, Green, Blue (RGB) to Hue, Saturation, Value (HSV) format. To confirm the reliability of HSV, we conducted 3 tests to make sure that the correct jewel gets flicked consistently. During the testing, the image was taken by the phone camera and the OpenCV was used for color conversion and saving of the image:

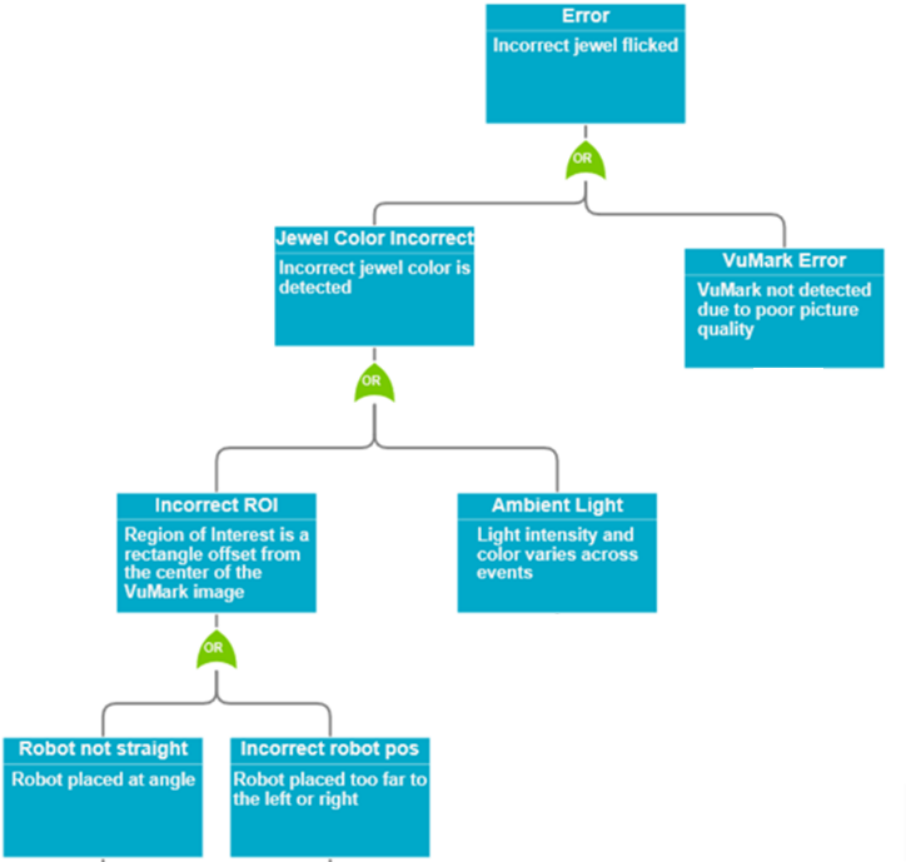
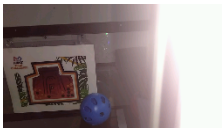
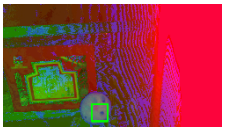
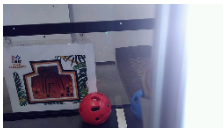
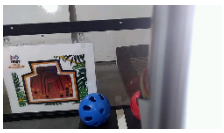


Figure 8: Jewel Flicker Fault Tree Analysis

In order to quantify the results of the previous test, we ran another set of tests. In the new tests, we changed the brightness of the ambient light and apply a color filter over the lights. The figure below

provides an example of the method where a green film covers the only light source besides the phone's flash (Experiment setup see Figure 9).

Table 8: Different Light Condition Testing for Jewel Flicker Missions

Light Condition	Image taken by Robot (camera flash is ON)	HSV image (shown as RGB)	Correct Jewel Flicked?
Dark			Yes
Normal			Yes
Very Bright			Yes

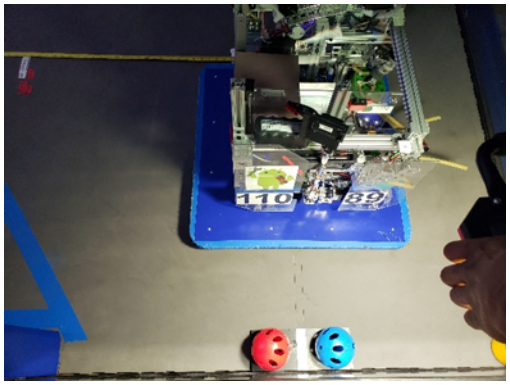


Figure 9: New Testing setup to test effects for the brightness of the ambient light and color filters

This table shows that regardless of the intensity or color of light, the algorithm accurately predicts the color of the jewel. Using HSV enables the robot to predict the color of the jewel with 100% accuracy.

Table 9: HSV Prediction Accuracy for Software Reliability

Trial	Luminosity (lx)	Description	Correct Jewel
1	180	Only flash	Yes
2	501	Standard conditions	Yes
3	2400	Light focused on VuMark	Yes
4	1100	Flash and focused green light	Yes
5	320	Flash and focused red light	Yes
6	95	Flash and focused blue light	Yes
7	1400	Standard and focused green light	Yes
8	450	Standard and focused red light	Yes
9	190	Standard and focused blue light	Yes

10	72-160	Pulsing light	Yes

4 Summary

This paper presented our approaches to improving software reliability in Android robots that integrates robot software and physical worlds together. Case studies were used to analyze fault trees, Software FMEA and the shortcut model for software reliability defect predictions in Android robots for the robot performance improvements in FTC robot competitions. The detail fault trees were analyzed to consider multiple factors including sensor reading and the environment uncertainties within Android robot and the field. With the evolvement of robot performances and capabilities, the software reliability theoretical model of the expected defect removal and software reliability growth model were generated based off the shortcut model for FTC robot software reliability characterizations. It was found that by comparing the software reliability model data and the robot performance in the field, the software reliability model is a very powerful tool to guide the team to close software reliability gaps in critical areas for consistent robot performances and also decide the robot redesign strategy and timing. Furthermore, the paper also presented the method to improve the software reliability through Android phone internal image process to minimize dependencies on the external physical sensor readings and the mechanism to mount the sensor. HSV based color processing methods were employed in order to ensure that prediction algorithms could accurately make decisions based on the input image data without being affected by the light conditions. All of these methods can be utilized in industries to improve software reliabilities when handling the physical world uncertainties and complicities.

Acknowledgments

Authors wish to thank reviewers, Keith Stobie and Kingsum Chow, for their invaluable assistance and discussions on software reliability. In addition, authors also wish to thank Oregon Robotics Tournament & Outreach Program (ORTOP) whose assistances are very crucial for the success of FTC events in Oregon.

References

[1] Mataric, Maja J., Koenig, Nathan and Feil-Seifer, David. 2007. Materials for Enabling Hands-On Robotics and STEM Education, In AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education, Stanford, CA.

[2] Chow, Kingsum, Chow, Ida, Niu, Vicki, Takla, Ethan and Brillhart, Danny. 2010. Software Quality Assurance in the Physical World. Proceedings of the Pacific Northwest Software Quality Conference. Portland, Oregon. USA.

[3] http://www.usfirst.org/ (accessed August 1, 2018).

[4]https://firstinspiresst01.blob.core.windows.net/ftc/game-manual-dw-part-2.pdf (accessed June 1, 2018).

[5] http://roboaztechs.org/technical-resources/ftc/ftc-robot-wiring-guide/ (accessed August 1, 2018).

[6] Musa, J.D. and Okumoto, K. 1984 A logarithmic Poisson execution time model for software reliability measurement. Proceedings of the 7*th* International Conference on software Engineering. 230-237, Orlando.

[7] Reifer, Donald J. 1979. Software Failure Modes and Effects Analysis, IEEE Transactions on Reliability. R-28 (3) 247-249.

[8] http://www.softrel.com/softwarereliability.pdf (accessed August 1, 2018).

[9] IEEE Standard Association, 2016, IEEE Recommended Practice on Software Reliability.

[10] http://www.fault-tree-analysis-software.com/fault-tree-analysis (accessed August 1, 2018).