**DESIGNING A SAFEKEEPING SYSTEM WITH HUMAN TOUCH DETECTION MODULE**

Leow Ken Hing Bryan

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Background And Purpose Of Research

The number of reported cases of burglary and property crimes in Singapore is getting increasingly rarer [1]. While this may be a cause for celebration, there are still many small and medium enterprises as well as independent shop owners who do not have even the most basic security system in their business premises. In a city renowned for being the easiest place to do business in [2][3], the number of such operations in Singapore will only increase over the next few years. As such, there is a need for an inexpensive and widely available safekeeping system which they can implement in their business premises to ensure the safety of valuable objects such as the cash register, safety deposit boxes and the like. This will not only help prevent theft, but also give these shop owners greater piece of mind knowing their possessions are secure.

Thus, the aim of our research project was to design an inexpensive, portable and reliable safekeeping system. For the purpose of fulfilling the first prerequisite, we have decided to utilise human touch detection combined with object motion activation, achieved by connecting a metallic conductor and a tilt sensor to an Arduino Leonardo microcontroller chip. Refer to Appendix A for the respective functions these sensors serve.

Since the program governing the safekeeping system will be written from scratch, we can conduct experiments to maximise the capabilities of our self-constructed sensor by manipulating the Arduino sketch on which it runs on, thereby augmenting its *Sensitivity, Consistency, Adaptability,* and *Reliability*, or *SCAR* for short.

2. Hypotheses:

**Hypothesis #1:** Using the relative difference between readings received from the capacitive sensor as the alarm trigger will improve SCAR as opposed to using a fixed threshold value.

**Hypothesis #2:** The lower the trigger value used, the higher the *Sensitivity* & the lower the *Consistency, Adaptability* and *Reliability* achieved.

3. Methods And Materials:

**Experiment #1** aimed to find out if the stimulus for the alarm trigger should be the RC time constant (i.e. the value received from the capacitive sensor) 1) exceeding a certain, pre-defined threshold value or 2) increasing by more than a certain value within a period of time (5 seconds). Separate sketches were written for both methods and they were each put through a series of tests that evaluated their *Sensitivity, Consistency, Reliability,* and *Adaptability* (*SCAR*). Refer to Appendix B for the 2 sketches. The *SCAR* evaluation criteria are detailed as follows:

*Sensitivity*:

* + This test evaluates the receptiveness of the sensor to an approaching hand. This is assessed by mainly two factors: the furthest possible distance (in cm) one can keep his/her hand from the sensor and still trigger the alarm, as well as how fast (in seconds) the sensor can respond to a hand.
  + A high *Sensitivity* rating would mean that the security system is better able to pre-empt an attempted burglary before it even happens.
  + To pass this test, the furthest possible distance should be more than zero on all trials.*\**

*Consistency:*

* This test evaluates how consistent the results gathered from the *Sensitivity* test are.
* Variation in *Sensitivity* should have been negligible for one to score high on this test.
* To pass this test, the number of successful alarms should be twice the number of failed alarms within the last 10 trials.*\**

*Reliability*:

* + This test takes into account any occurrences of false alarms – alarms that are triggered spontaneously when they are not intended or supposed to. An alarm should be as ***reliable* & *consistent*** as possible in order to prevent false alarms and/or malfunctions.
  + To pass this test, the number of successful alarms should be at least twice the number of false alarms.*\**
* *Adaptability*:
  + This test evaluates how adept a sensor is to changing environments and situations.
  + The sensor will be put through a variety of environments and situations, and the results from each will be compared and compiled to give a final score. A high adaptability score requires the sensor to fare decently across most or all surface types.
  + The idea behind such a test is to determine the versatility of the sensor & in turn our security system. It is important to ensure that the system can function under almost any environment it is put into, so that users and consumers do not have to make manual alterations to the program to adapt it to their needs.

*\** A pass automatically grants a score of 3 and above (does not apply to Adaptability tests)

The following describes the actual methodology used involving the *SCAR* criteria:

1. Turn on the security system and allow the capacitive sensor readings to stabilise.
2. Conduct the *Sensitivity* test:
   1. Move your hand gradually towards the capacitive sensor (at approx. 4 cm/s) until the alarm is triggered.
   2. Record the distance (in cm) between your hand and the capacitive sensor.
   3. Remove your hand for the readings to return to normal.
   4. Next, hover your hand just 2cm over the sensor and start a stopwatch. When the alarm triggers, stop and record the time.
   5. Repeat steps 2a–2d twice and calculate the average distance & time.
3. The final *Sensitivity* score (out of a possible 10) will be determined as follows:

Where distance (cm) is adjusted for the passing mark of 3, and time (s) is moderated by 0.2s (the maximum ideal reaction time of the capacitive sensor).

1. Calculate the *Consistency* score:

Where Score S max is the highest *Sensitivity* score attained and Score S min is the lowest.

The maximum possible variation is being tabulated and subtracted off 10.

1. Determine the sensor's *Reliability*:
   1. Calculate the percentage of intended alarms over total number of alarms sounded.
2. Conduct the *Adaptability* test:
   1. Put the sensor through the *SCR* tests in the control (default) environment.\*\*
   2. Relocate it to simulate different environments and situations, such as being placed on glass surfaces, being covered by plastic/thick paper, surrounded by metal objects etc. Conduct *SCR* tests each time the environment changes and benchmark the scores against those taken in the control environment.
   3. Tabulate the results and determine the average performance across all surfaces.

\*\* The control environment is an empty plastic tabletop. This is used for all SCR tests.

After determining the method of alarm trigger, we then had to determine the optimal value to use in our program. **Experiment #2** entailed conducting the same *SCAR* tests on a wide range of trigger values and plotting the change in rating across these values on line graphs. The evaluation criteria and methodology are identical to that of Experiment #1.

Results & Discussion:

*For the full & finalised Arduino sketch used for the security system, refer to Appendix C below.*

Experiment #1 :Evaluating the better method of alarm trigger

**Method 1** utilises the pre-defined threshold value while **Method 2** detects if the increase is equal to or greater than 15 within a 5 second period. **Method 2** surpassed **Method 1**'s *Consistency* score of 7.3 with an impressive 9, but is slightly less sensitive as a trade-off. However, such a minor drawback is nothing compared to the abysmal performance in *Adaptability & Reliability* witnessed in **Method 1**, which barely scraped a pass in the former with a 32% false alarm rate.

Figure 1. *SCAR* analysis on two different trigger methods

This proves our hypothesis – **Method 2** is overall the better option compared to **Method 1.** The extra *Sensitivity* that **Method 1** can offer will not matter if it is so prone to false alarms, since that would make the entire system capricious in nature to begin with. In addition, its low *Adaptability* score is indicative that the high *S & C* scores are only contextual, and that it cannot provide the same level of proficiency in its sensing capabilities once the environment changes.

Experiment #1 Discussion

Initially in **Method 1**, even the slightest increase caused by an approaching hand would result in the exceeding of the threshold value, thereby causing a high *Sensitivity* rating*.* However, this selfsame feature is the exact reason why it has a much lower *A & R* rating – the RC time constant naturally increases and decreases on its own and occasionally, this increase would be enough to surpass the threshold value, setting off a false alarm. In addition, it should be noted that the grounding of the Arduino is not perfect. Hence, the RC time constant is sometimes unable to decrease back to its original value, resulting in the creation of a new, higher mean. This would have a devastating effect on the system as it will cause the system to malfunction, necessitating a modification of the threshold value for the sensor to be operational again. In essence, a hard-coded threshold value was too resistant to change to be of any practical, long-term use.

**Method 2**, on the other hand, was much more adaptable to change because the threshold value was based off the mean, making the program a lot more flexible. Its *Consistency* was also noticeably higher as it could compensate for the times when the sensor is unable to discharge fully, as a result of imperfect grounding. As a trade-off, the sensor seems to have lost some *Sensitivity*, but this can be countered by adjusting the difference threshold (15 in this case) to a smaller value. More of this is covered in Experiment #2.

Experiment #2 :Determining the optimal difference threshold

Similarly, another *SCAR* analysis was conducted to determine the optimal difference threshold to use in our program, and the results are shown in Fig. 2

Figure 2. *SCAR* analysis to determine optimal difference threshold

From the graph, a couple of trends can be observed:

* **Sensitivity**decreases as the Difference Threshold Value (DTV) increases, and this decrease is more significant in the first half of the graph, when the DTV is still relatively small. The graph approaches a plateau as the DTV goes beyond 30. This agrees with our hypothesis.

Initially, a small increase in DTV would result in a huge decrease in distance between the hand and the sensor. However, as the DTV became increasingly large, it soon came to a point where distance no longer had to decrease, and instead more *time* was needed for enough charges to be drawn away from the sensor. This time difference was relatively small, hence the values started to agglomerate at the 3.5 mark from 24 DTV onwards. Despite so, all values passed the test, automatically warranting them a score of 3 and up.

* ***Consistency & Reliability*** generally saw extremely excellent performances across the board. The former averaged at 9.2 out of 10, and 76% of DTVs achieved a perfect *Reliability* rating. While the initial *Reliability* scores were noticeably sub-par at the start, they rose quickly to peak at 10. *Reliability*'s trend agreed with our hypothesis, while that of *Consistency* does not, since there is no overall general trend observed with the latter.

This can be attributed to the versatile nature of our program, which could easily handle any spontaneous changes in the mean RC time constant by updating the two locally stored values in real-time, albeit at different frequencies. This soft-coded approach makes for a very trustworthy and stable alarm system.

* ***Adaptability*** witnessed the most interesting trend and did not follow our hypothesis. Peaking at 18 DTV, it formed a bell curve with relatively steep gradients, the right side being steeper than the left. At 26 DTV, the graph approached a plateau and hovered around the 1/10 mark.

This is because the environment in which the sensor is placed in will affect both the degree of natural variation as well as the degree of increase in the RC time constant. When the DTV is too high, its *Sensitivity* rating would be severely impacted when put into situations where the increase in RC time constant starts to fall short of the required increase, such as being covered by plastic and thick paper. On the other hand, a low DTV might obtain a high *Sensitivity* rating but it will come at the expanse of *Reliability* when put into an environment with high RC time constant fluctuations, such as being surrounded by metal objects. This causes both high and low DTV values to suffer in terms of *Adaptability*, resulting in the steep bell curve seen. A trade-off between *Sensitivity* and *Reliability* is imperative to achieving optimal *Adaptability*, and 18 DTV seems to offer this ideal combination.

However, it should be noted that this in no way signifies that 18 DTV is the "best" value among all the DTVs tested; it simply means that it is the most universal. Going back to Fig. 2, it is clearly seen that there are other values, such as 12 & 14, that offer more *Sensitivity* than that of 18, and yet still possess the same level of *Reliability* and *Consistency* that 18 does.

As such, user discretion is advised in determining the best value to use. If the system is to be deployed in various locations where the conditions can differ greatly, a DTV of 16-20 would be recommended. Conversely, if the system is largely going to remain in one place, then a DTV of a higher *Sensitivity* can be used, such as 14, 12 or even 10. Note however that the *SCR* tests conducted in Experiment #2 took place on an empty tabletop made of plastic (the control environment used for all tests). Surfaces of different materials, especially that of metal, might have varying results from that portrayed in Fig. 2. In such a case, leaning towards 18 DTV might be a safer option.

Conclusion

This project presents the possibility of using capacitive sensing coupled with a motion sensing device to create a highly reliable, affordable and portable security system that can safeguard an object against unwanted burglaries or theft. The combination of both sensors makes for a highly trustworthy safekeeping system that allows relevant parties to be notified of the nature of the situation simply by recognising the different alarm protocols that correspond to the different circumstances. Such a safekeeping system is as affordable as it is reliable, and can be assembled from scratch with a mere hundred dollar budget or less. Its compact and space-efficient design also makes it a highly portable security system. Independent business operators would benefit greatly from this system, since it offers a low-cost solution to managing the safety of their possessions within their business premises.

Future Works

This project has presented its fair share of improvements and innovations to counter the problem of property safekeeping, and while extensive research has already been conducted to optimise the *SCAR* of the program, it nonetheless offers huge potential in opening up a myriad of future research opportunities to augment the said qualities of this safekeeping system, as well as improve its overall user-friendliness. These include, but are not limited to: automatically switching the Difference Threshold Value (DTV) to that of a higher value when the natural variation in the RC time constant increases (and vice versa), an in-depth exploration at the relationship between the *stability* and *sensitivity* of the sensor in regards to the RC time constant, developing a device that allows the owner to remotely control the system by turning it on and off, and possibly integrating an in-built camera into the system that will automatically take a picture of the offender and store it within a retrievable databank along with other relevant information such as date and time.

~•~

References

[1] Moore, D. and Sciera, B. InterNations.org. (2012). "*Safety, Law, and Crime in Singapore*". Retrieved from https://www.internations.org/singapore-expats/guide/16087-safety-security/safety-law-and-crime-in-singapore-16092 .

[2] Holliday, K. CNBC News. (2014). "*Singapore easiest place to do business for 9th year*". Retrieved from http://www.cnbc.com/2014/10/29/singapore-easiest-place-to-do-business-for-9th-year.html .

[3] Ng, J. Y. TODAY online News. (2015). "*Singapore easiest place to do business 10 years in a row*". Retrieved from http://www.todayonline.com/business/singapore-still-easiest-city-do-business-world-bank .

[4] Singapore SME Directory. Retrieved December 2015 from http://www.singapore-sme.com/ .

APPENDIX A | The 2 States of Emergency (SoE)

* State of Emergency 1 (SoE 1):
  + A person is in the vicinity of the secured object and about to come into contact with it. Depending on the context and nature of the situation, the offender may or may not harbour motives of theft or burglary. Warning siren goes off and the LED blinks at regular intervals to either notify the offender to retreat or alert relevant parties of an intruder, depending on when and where this system is being deployed e.g. at a warehouse, at a museum during operating hours, at a convenience store etc. The LCD will display text to ensure that concerned parties are notified of the issue. This is the first layer of protection, realised using the capacitive sensor as pictured in Figure 3

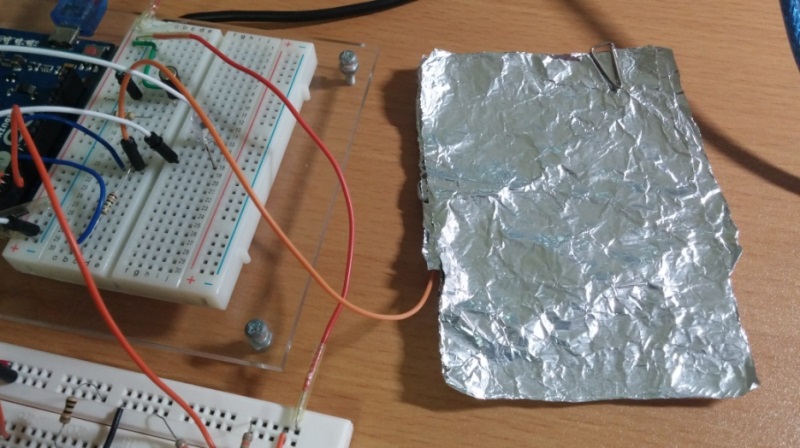


Figure 3: Touch sensor setup in the safekeeping system

* State of Emergency 2 (SoE 2):
  + The object has been displaced from its original position, most certainly indicating intentional theft. High-pitched alarm siren goes off, the LED blinks at regular, shorter intervals, and the tri-colour LED is also activated, flashing a random colour at regular intervals. The LCD will display text to notify all parties of the theft. This is the second layer of protection, realised using the tilt sensor.



Figure 2: Tilt sensor setup in the safekeeping system

APPENDIX B | The 2 Arduino Sketches Used For Experiment #1

Method 1: exceeding a certain, pre-defined threshold value

|  |
| --- |
| **#include <CapacitiveSensor.h>**  Pins 6 & 7 are the send and receive pins respectively.  **CapacitiveSensor capSensor\_6\_7 = CapacitiveSensor (6,7);**  **void loop () {**  **long value = capSensor\_6\_7. capacitiveSensor(30);** // read from the sensor  **if (value>70) { }**  } |

Method 2: increasing by more than a certain value within a period of time (5 seconds)

|  |
| --- |
| **#include <CapacitiveSensor.h>**  Pins 6 & 7 are the send and receive pins respectively.  **CapacitiveSensor capSensor\_6\_7 = CapacitiveSensor (6,7);**  **elapsedMillis CapStimer;**  **#define CapSinterval 5000**  Measures the time (in ms) since the program started or the timer was reset.  **int oldValue;**  **int newValue;**  Some parts of the program were cut since they were irrelevant to the purpose of Appendix B. For the full sketch used, refer to Appendix C.  **void loop() {**  **average = total / numReadings;**  newValue is updated in real-time and takes on the average of the past 10 capacitive sensor readings.  If 5000 ms has passed...  ...reset CapStimer by subtracting CapSintveral  ...and let oldValue take on the value of average  **newValue = average;**  **if (CapStimer > CapSinterval) {**  **CapStimer -= CapSinterval;**  **oldValue = average;**  **}**  **if ( ((millis()) >= 5100) && ((newValue - oldValue) >= 18) ){ }**  } |

APPENDIX C | The Full Arduino Sketch Used For The Program

|  |
| --- |
| **#include <LiquidCrystal.h>**  Defines the libraries to use, for the LCD, timer and capacitive sensor respectively.  **#include <elapsedMillis.h>**  **#include <CapacitiveSensor.h>**  A double slash ( // ) instructs the Arduino to ignore any subsequent text in that line.  **// == Capacitive Sensor Variables ==**  Pins 6 & 7 are the send and receive pins respectively.  **CapacitiveSensor capSensor\_6\_7 = CapacitiveSensor(6,7);**  **elapsedMillis CapStimer;**  Measures the time (in ms) since the program started or the timer was reset.  **#define CapSinterval 5000**  **int oldValue;**  Declares two variables with type integer (int)  **int newValue;**  **const int numReadings = 10;**  **int readings[numReadings];**  **int readIndex = 0;** // The index of the current reading  **int total = 0;** // The running total  **int average = 0;** // The average  **int storedValue;**  **// == LCD Variables ==**  **LiquidCrystal lcd(13,12,11,10,9,8);** // The pins used for the LCD.  **// == LED Variables ==**  **const int LEDPin = A2;**  Frequency of LED blinking in *State of Emergency 1 & 2* is defined as 500 ms and 100 ms respectively. Refer back to Appendix A for details on the different *SoEs*.  **elapsedMillis timerLED;**  **#define intervalLED1 500**  **#define intervalLED2 100**  **// == Music Variables ==**  **const int SpeakerPin = 5;** // The speaker is connected to Pin 5  **int note;**  Both are used in a **SWITCH CASE** function to alternate playing two notes of different frequencies at regular intervals.  **int note2;**  **elapsedMillis timerMUSIC;**  **#define intervalMUSIC1 500**  **#define intervalMUSIC2 100**  **// == Tri-colour LED Variables ==**  Pins 4, 3 & 2 control the red, green and blue colour of the tri-colour LED respectively.  **const int ledPinR = 4;**  **const int ledPinG = 3;**  **const int ledPinB = 2;**  **int brightR;**  These variables can either be 0 or 1, and they control the state of each colour on the tri-colour LED.  **int brightG;**  **int brightB;**  **int triLEDrepeat;**  **elapsedMillis timerTriLED;**  **#define intervalTriLED 50**  **// == Tilt Sensor Variables ==**  Both states are initiated at LOW to prevent false alarms.  **int currentState = LOW;**  **int previousState = LOW;**  **int TSPin = A0;** // The tilt sensor is connected to Pin A0  **// == Timer Variables ==**  **elapsedMillis timerAlarm;**  **elapsedMillis timerAlarm2;**  Data is transferred from the Arduino board to the Serial Monitor at a baud rate of 9600.  **void setup() {**  **Serial.begin(9600);**    **// ~~ Capacitive Sensor Set-up ~~**  **CapStimer = 0;** // Resets cap sensor's elapsedMillis timer  **// ~~ LCD Set-up ~~**  Defines LCD display as 16 columns x 2 rows  **lcd.begin(16,2);**  **lcd.print("Object secure");**  Text printed under "Default" condition. This can be modified to print user-defined phrases.  **// ~~ LED Set-up ~~**  **pinMode(LEDPin, OUTPUT);** // Defines the LED pin as an output pin  **digitalWrite(LEDPin, LOW);** // Sets the LED in its OFF state  **timerLED = 0;** // Resets LED's elapsedMillis timer  **// ~~ Music Set-up ~~**  **note = 0;**  // Value of 0 represents first note played for *State of Emergency 1*  **note2 = 0;** // Value of 0 represents first note played for *State of Emergency 2*  **timerMUSIC = 0;**  **// ~~ Tri-colour LED Set-up ~~**  **pinMode(ledPinR, OUTPUT);**  **pinMode(ledPinG, OUTPUT);**  **pinMode(ledPinB, OUTPUT);**  **timerTriLED = 0;**  **// ~~ Tilt Sensor Set-up ~~**  **pinMode(TSPin, INPUT);**  //Instructs TSPin to read the current received from the tilt sensor  **// ~~ Timer Set-up ~~**  **timerAlarm = 0;**  **timerAlarm2 = 0;**  **} // END SET-UP**  **void loop() {**  // **FINDING AVERAGE CAPSENSOR READING**: (over arrays of **10**)  **total = total - readings[readIndex];** // subtract the last reading  **readings[readIndex] = capSensor\_6\_7.capacitiveSensor(30);** // read from the sensor  **total = total + readings[readIndex];** // add the reading to the total  **readIndex = readIndex + 1;** // advance to the next position in the array  **if (readIndex >= numReadings) {**  // if the end of the array is reached…  **readIndex = 0; }** // …wrap around to the beginning  **average = total / numReadings;** // calculate the average (divide by 10)  newValue is updated in real-time and takes on the average of the past 10 capacitive sensor readings.  If 5000 ms has passed...  ...reset CapStimer by subtracting CapSintveral  ...and let oldValue take on the value of average  **newValue = average;**  **if (CapStimer > CapSinterval) {**  **CapStimer -= CapSinterval;**  **oldValue = average;**  **}**  **Serial.println(oldValue);**  **|** Print both variables in the Serial Monitor.  **Serial.print("\t");**  If the DTV is >= 18, the following will be executed as the alarm protocol for *State of Emergency 1.*  Note: this can only function if the program has been running for more than 5 seconds (millis()>=5100), since oldValue will only update with its first value 5 seconds into the program's runtime.  **Serial.println(newValue);**    **// C A P A C I T I V E S E N S O R | *State of Emergency 1***  **if ( ((millis()) >= 5100) && ((newValue - oldValue) >= 18) ) {**  **timerLED = 0;**  Resets all elapsedMillis timers to 0 to ensure the timings are accurate. Large values (>2x the interval) are known to cause breakdowns in the program.  **timerMUSIC = 0;**  **timerAlarm = 0;**  **timerAlarm2 = 0;**  **timerTriLED = 0;**    **// When object touched | LCD**  **lcd.home();** // Bring cursor to home / column 0, row 0 i.e. (0,0)  **lcd.print("WARNING: ");**  Text printed under *SoE 1*. This can be modified to print user-defined phrases.  **lcd.setCursor(0,1);**  **lcd.print("Intruder Alert! ");**  **while (timerAlarm <= 3500) {** // for 3.5 seconds…  **// When object touched | LED**  **if (timerLED > intervalLED1) {** // after 500 ms…  **timerLED -= intervalLED1;** // …reset timer  **digitalWrite(LEDPin, !digitalRead(LEDPin)); }** // …and invert state of LED.    **// When object touched | Music**  **switch(note)**  **{**  **case 0:**  The **SWITCH CASE** function. Execute this statement when note = 0 (initially true). The **break** keyword causes the program to jump out from the **switch** statement.  **tone (SpeakerPin,262,250);**  **break;**  **case 1:**  **tone (SpeakerPin,310,250);**  **break;**  **}**  **if (timerMUSIC >= intervalMUSIC1)** { // after 500 ms…  **timerMUSIC -= intervalMUSIC1;**  // …reset the timer  // …and invert the value of **note** in Boolean operators, i.e. convert 0 to 1 and vice versa. This causes the other case to become true, executing their statements. When put in a loop, this effectively toggles the speaker between two frequencies at regular intervals.  **note = !note; }**  **} // CLOSING "WHILE" FUNCTION**    **lcd.clear();**  **lcd.home();**  Text printed under Post-*SoE 1* condition. This can be modified to print user-defined phrases.  **lcd.print("Intruder alarm");**  **lcd.setCursor(0,1);**  **lcd.print("was sounded.");**  Hard-coded reset to ensure the LED will be off after the alarm protocol is finished.  **digitalWrite(LEDPin, LOW);**  **} // CLOSING *SoE 1***  **// T I L T S E N S O R**  **currentState = digitalRead(TSPin);** // Read value of incoming current.  **if ( (currentState == HIGH) && (previousState == LOW) ) {**  **timerLED = 0;**  Resets all elapsedMillis timers to 0 to ensure the timings are accurate. Large values (>2x the interval) are known to cause breakdowns in the program.  **timerMUSIC = 0;**  **timerAlarm = 0;**  **timerAlarm2 = 0;**  **timerTriLED = 0;**  **while (timerAlarm2 <= 5000) {**    **// When object moved | LCD**  **lcd.home();**  **lcd.print("WARNING!!! ");**  Text printed under *SoE 2* condition. This can be modified to print user-defined phrases.  **lcd.setCursor(0,1);**  **lcd.print("Object stolen!! ");**  **// When object moved | LED**  **if (timerLED > intervalLED2) {** // after 100 ms…  **timerLED -= intervalLED2;**  //…reset the timer  **digitalWrite(LEDPin, !digitalRead(LEDPin)); }** // …and invert state of LED.  **// When object moved | Music**  **switch(note2) {**  **case 0:**  **tone (SpeakerPin, 3000,250);**  **break;**  **case 1:**  **tone (SpeakerPin, 2000, 250);**  **break;**  **}**  **if (timerMUSIC >= intervalMUSIC2) {**  **timerMUSIC -= intervalMUSIC2;**  **note2 = !note2;**  **}**  **// When object moved | Tri-colour LED**  **brightR=random(2);**  Every 50 ms, brightR, brightG and brightB will be randomised to represent either 0 or 1 to control the tri-colour LED, resulting in auto colour mixing.  **brightG=random(2);**  **brightB=random(2);**  **if (timerTriLED >= intervalTriLED)**  **{**  **timerTriLED -= intervalTriLED;**  **digitalWrite(ledPinR, brightR);**  **digitalWrite(ledPinG, brightG);**  **digitalWrite(ledPinB, brightB);**  **}**    **} // CLOSING "WHILE" LOOP**  **digitalWrite(LEDPin, LOW);**  Hard-coded reset to ensure the tri-colour LED will be completely off after the alarm protocol is finished.  **digitalWrite(ledPinR, LOW);**  **digitalWrite(ledPinG, LOW);**  **digitalWrite(ledPinB, LOW);**  **lcd.clear();**  **lcd.home();**  **lcd.print("WARNING: OBJECT");**  Text printed under Post-*State of Emergency 2* condition. This can be modified to print user-defined phrases.  **lcd.setCursor(0,1);**  **lcd.print("WAS STOLEN!!");**  **} // CLOSING "IF" FUNCTION**  Move **currentState** to **previousState** to prepare for the next cycle.  **previousState = currentState;**  **delay(50);** // 50 ms buffer period  **} // END LOOP** |