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

Leow Ken Hing Bryan1, Arun Shankar Narayanan2, Tan Kok Kiong2,

1Dunman High School (Secondary), 10 Tanjong Rhu Rd, Singapore 436895

2National University of Singapore, 21 Lower Kent Ridge Rd, 119077

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

Abstract:

Existing security systems often make use of sensors and detectors placed around the vicinity of a secured object to pre-empt an attempted robbery before it happens. Such an alarm system can be expensive, since the components required to assemble such a system may not be cheap and can incur considerable operating costs in the long run. As such, we have devised an inexpensive and affordable safekeeping system that utilises human touch detection and/or object motion activation to trigger an alarm. Such a system leverages on capacitive sensing, a technology based on capacitive coupling that takes the human body capacitance as input – that is, the property of the human body that enables it to act as a capacitor. It also utilises a tilt sensor device to detect changes in the absolute position of the object. When used in conjunction with an already established defence matrix to safeguard the premises, it can add an extra layer of security and make intruder detection unavoidable. It can also function as a standalone security system to safeguard an object rather than an area, which would be extremely affordable as it incurs minimal operating and maintenance costs.

Introduction:

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 (SMEs) as well as independent shop owners who do not have even the most basic security system in their business premises. Therefore, there is still the possibility of shop theft occurring in these locations which either goes unnoticed or unreported. In a city renowned for being the easiest place to do business in [2][3], the number of SMEs and independent shop owners in Singapore will only increase over the next few years.

As such, there is a need for an inexpensive safekeeping system that is widely available to all independent business operators, for implementation in the premises of their businesses 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 since they cannot be watching over the safety of their possessions all the time. It is especially helpful for those who have to single-handedly manage their business operations without another partner physically present, such as in the case of small branches, independent businesses and convenience stores commonly found in malls and community hubs around Singapore [4].

Alternatively, this system can also be used in conjunction with an existing safekeeping system to guard a high-value object. By allowing this system to operate separately from the one that is already in place, it can serve as an extra layer of security and provide backup in the case where the offender manages to bypass one or more of the other sensors, thereby increasing reliability of the defence as a whole.

All these benefits come at an extremely affordable price. The entire system will cost less than S$100 to set-up if one were to purchase all the components from scratch. In addition, the maintenance and operating costs are also extremely low as the system can be powered by a simple 9V battery [5], and the components are easily replaceable in the event that they stop working. This makes it a cheap and effective security system in the long run.

Methods And Materials:

A typical security alarm employs the following components [6]:

1. Alarm Control Panel (ACP):

The "brain" of the system which reads sensor inputs, tracks arm/disarm status, and signals intrusions. This is typically one or more computer circuit boards inside a metal enclosure, along with a power supply.

However, this particular safekeeping system we have devised utilises an Arduino Leonardo chip as the ACP, making it highly space-efficient and widely available since it can easily be obtained through online ordering or otherwise. Its compact design also allows it to be placed alongside the guarded object in a concealed manner.

1. Sensors:

These are the devices, which detect intrusions and attempts of burglary. In conventional premise safekeeping systems, they are typically placed around the perimeter of the protected area and detect intruders by a variety of methods, such as monitoring doors and windows for opening or interiors for motions, sound, vibration and other disturbances.

In this safekeeping system, two types of sensors are employed: a capacitive sensor and a tilt sensor which are placed over and on the object respectively. The former will detect an incoming human touch, while the latter will detect changes in the object's position.

1. Alerting devices:

These indicate an alarm condition, and serve the dual purposes of warning the relevant parties of an intrusion, and potentially scaring off burglars.

In this safekeeping system, there are four of such devices, namely a light-emitting diode (LED), a liquid crystal display (LCD), a speaker and a tri-colour LED; and their use

corresponds to different *states of emergency.*

States of emergency:

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.
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.

Capacitive Sensor

With the help of a touch-sensing software library developed by Paul Badger, we can transform any 8-bit Arduino board into a touch-activated ACP. This capacitive touch sensing software library allows the Arduino to detect human touch by monitoring the charge/discharge timing cycle of an RC (resistor-capacitor) network, formed by a single resistor and the touch electrode capacitance [7]. Pictured in Fig. 1 is the touch sensor used in the safekeeping system. The triggering of this sensor would be associated with *State of Emergency 1.* The touch electrode capacitance required in capacitive sensing can actually be provided only in the form of a simple jumper wire. However, we have decided to attach a piece of aluminium foil to the end of this (orange) wire, secured using a metallic paper clip. This will not only function as a proper touch sensor, but also helps to spread the charges throughout the foil plane, increasing its sensitivity.

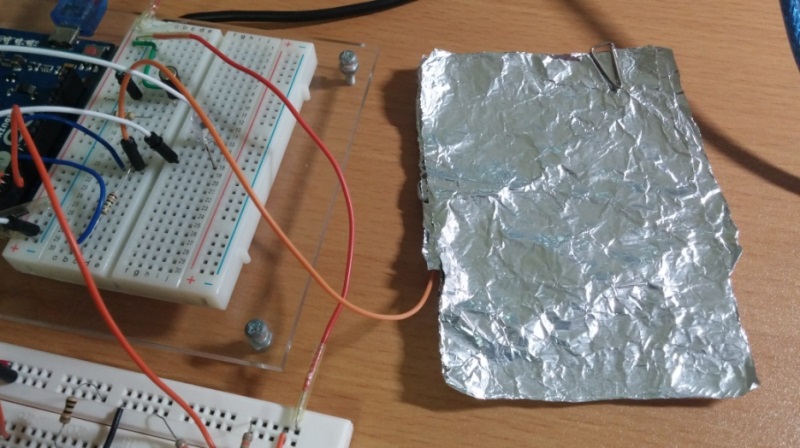


Figure 1: Touch sensor setup in the safekeeping system

***Working principle behind capacitive sensing:***

* 1. The software first toggles the send pin to a new state, and then waits for the receive pin to change to the same state. The delay between the send pin changing and the receive pin changing is determined by an RC time constant, which can be represented by the following equation:

(1)

Where R is the value of the resistor and C is the capacitance of the receive pin (Cpin) combined with the capacitance introduced at the touch plate by the interfacing object (i.e. the hand of an approaching person) (Ctouch).

* 1. In our safekeeping system, the value of R is constant at 1x106 since a 1M Ω resistor was used. The value of Cpin will also be a fixed value for the same receive pin used, however it must be kept as low as possible to ensure touch detection which is a variation of only a few picofarads (typically 5pF) [8].
  2. Under the absence of a touch input, Ctouch will approach zero in the ideal situation where it can effectively discharge with good grounding. The board can be grounded through one of the following ways:
     + Connecting the mains supply to the laptop which is then connected to the Arduino
     + Connecting the Arduino ground to an earth ground e.g. a water pipe [5].
  3. Once the RC time constant is met and both pins are at the same state, the send pin will be toggled again and the cycle repeats. This causes the receive pin to be periodically charged and discharged through the fixed resistor, the frequency of which is dependent on the time constant.
  4. When a touch from an electrical conductor (such as a metal object or a hand) is introduced at the touch plate, touch capacitance Ctouch will increase. This initially minute increase will be amplified by a million times since C is multiplied by R (1x106), resulting in a noticeable increase in the RC time constant.
  5. This variation in the RC timing due to the electrode capacity change is detected and eventually reported to the Arduino board, which can trigger further actions if it satisfies a condition as defined by the user.

During the development process, it was debated whether the stimulus for the alarm trigger should be the RC time constant 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*). The collected data will be presented in the Results section, under the subsection Experiment CS01. The evaluation criteria used for the *SCAR* tests are detailed as follows:

* *Sensitivity*:
  + This test evaluates the receptiveness of the sensor to an approaching hand. This is evaluated 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 hands which approach at different speeds.
  + The further the distance / the shorter the time, the more sensitive the sensor is said to be. 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. This is related to the *consistency* test, as both are evaluating how much the system can be trusted.
  + An alarm should be as ***reliable* and *consistent*** as possible in order to prevent false alarms and/or malfunctions, both of which may lead to negative consequences.
  + To pass this test, the number of successful alarms should be thrice the number of false alarms within the last 10 trials.*\**
* *Adaptability*:
  + This test evaluates how adept a sensor is to changing environments and situations.
  + The sensor will first go through the *SCR* tests in the control (default) environment.\*\* After which, the sensor will be put through a variety of environments and situations, such as being placed on glass/wood surfaces, being covered by plastic/thick paper, surrounded by metal objects etc, going through the same *SCR* tests each time the environment changes. The results will be compared and compiled to give a final score. A high adaptability score would require 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 control environment is an empty tabletop made of plastic. 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. If it was too large, the increase in the RC time constant may fall short of this threshold and fail to sound the alarm when needed. Conversely, too small a value might cause a false alarm to be triggered due to the natural variation in the RC time constant. As such, finding the optimal value would be imperative in making our security system as *Sensitive, Consistent, Reliable & Adaptable* as possible. To do this, *SCAR* tests were conducted on a wide range of values and plotted the change in rating across these values on line graphs. The collected data will be presented in the Results section, under the subsection Experiment CS02.

Tilt Sensor

A tilt sensor is a type of switch which makes or breaks contact when tilted at a certain angle. It is a cylindrical object that stores a free-moving metal ball. [5] Two non-polarised lead pin protrusions can be found on the tilt sensor which will be connected to the 5V Arduino power supply and an Arduino pin respectively, as seen in Fig. 2. The right pin is connected to the power supply and the left pin is connected to Pin A0 on the Arduino board via the green jumper wire.When the ball is displaced from the lead pins due to slight disturbances, a momentary break in contact with the two lead pins will occur and no current will be received by the Arduino pin as a result. A variable can be instructed to read the value of the current in real-time and compare it to previously stored values of an earlier time frame. If the previous value is "LOW" and the real-time value is "HIGH", it will allow us to conclude that a connection was re-established, indicating that it was moved. This can in turn trigger a series of alarms. The triggering of this sensor would be associated with *State of Emergency 2.* Refer to the sections of code in the full Arduino sketch (under the **Appendix**) that pertain to the tilt sensor.



Figure 2: Tilt sensor setup in the safekeeping system

Liquid Crystal Display (LCD)

The main purpose of the LCD is to alert relevant parties of an intruder, in the case where the alarm is not seen or heard.

The process of controlling the LCD involves putting the data that form the desired image into the data registers, then inputting instructions into the instruction register to convert them into text. The LiquidCrystal Library automates this conversion process, allowing us to input text directly into the Arduino IDE to be converted into LCD data. [5] Table 2 shows a list of possible phrases that will be printed on the LCD in varying situations.

Table 2. LCD text according to different conditions

|  |  |
| --- | --- |
| **Condition / State** | **Text on LCD** |
| Default | "Object secure" |
| *State of Emergency 1* | "WARNING: Intruder Alert!" |
| Post-*State of Emergency 1* | "Intruder alarm was sounded." |
| *State of Emergency 2* | "WARNING!!! Object stolen!!" |
| Post-*State of Emergency 2* | "WARNING: OBJECT WAS STOLEN!!" |

If one wishes to make the LCD display other phrases, he/she can simply modify the program by inputting the desired phrase directly into the program. Refer to the **Appendix** for the full Arduino sketch, and the appropriate locations to insert customised phrases.

Alarms | Tri-Colour Light-Emitting Diode (Tri-Colour LED)

The bulb of the tri-colour LED contains the three primary colours – red, green and blue, and each

can be independently turned on or off. By mixing and matching these 3 colours, we can create a total of 7 colours – red, green, blue, purple, lime, cyan and white, as pictured in Fig. 3below**:**

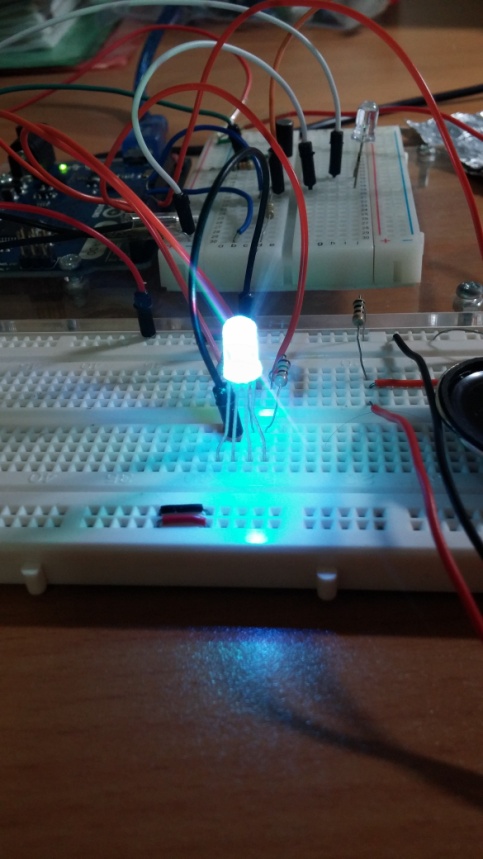
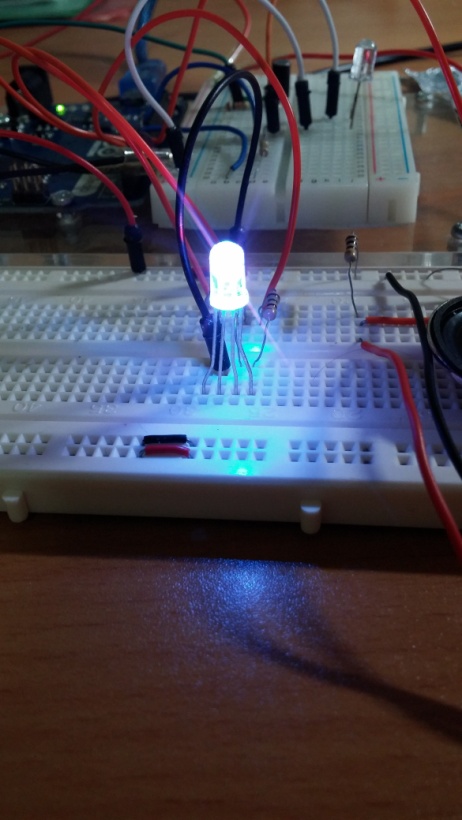
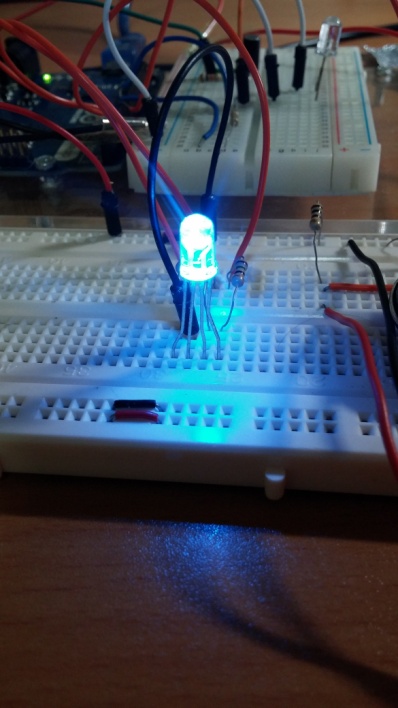
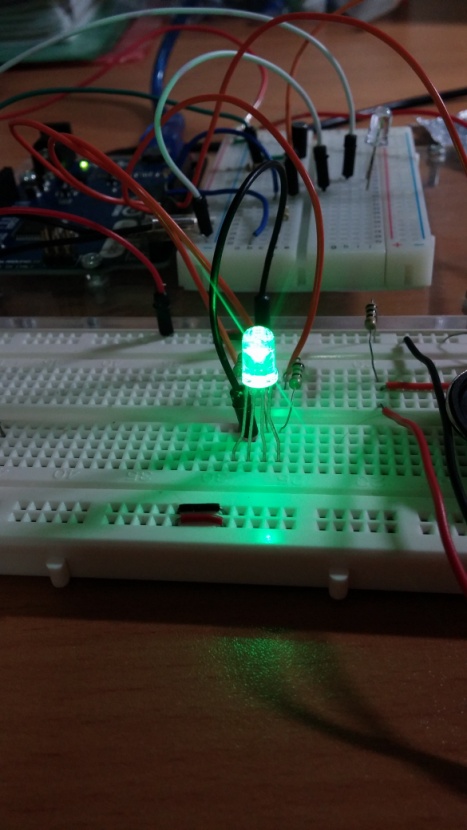
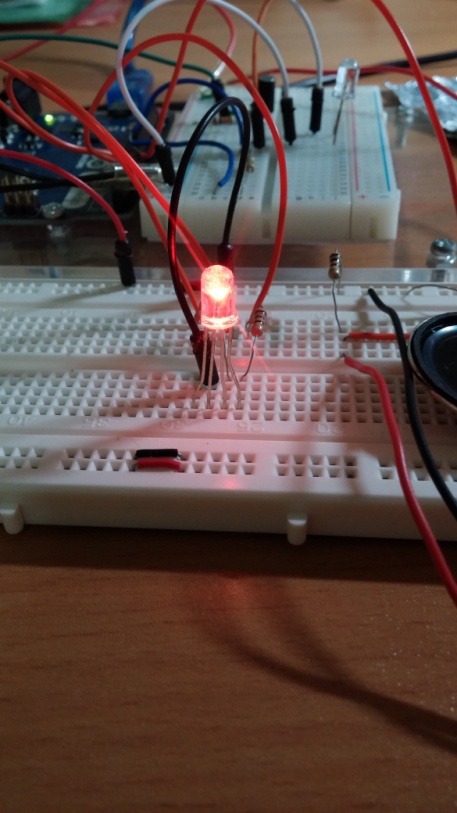


Figure 3. Different colours displayed using the tricolour LED

In our safekeeping system, the colour mixing will be done automatically using the random() function, in which the Arduino board will generate a random number to assign to the specified term, taking any integer below the number inputted in the parenthesis. In this case, "brightR", "brightG" and "brightB" will either be 0 or 1, and their outcome will in turn affect the state of their respective colour. This is repeated every 50 ms and as a result, the tri-colour LED will continuously change colour. Refer to the sections of code in the full Arduino sketch (under the **Appendix**) that pertain to the tri-colour LED for more details.

Alarms | Light-Emitting Diode (LED), Tri-Colour LED, Speaker

These form the alarm mechanism of our safekeeping system and serve the primary purpose of alerting nearby parties of a burglar. Their behaviour changes with the *State of Emergency* to

reflect different urgency levels, and also to allow relevant parties to differentiate the possible situations if they are unable to witness the theft as it occurs.

1. *State of Emergency 1*

* LED blinks on and off every 500 ms.
* Speaker will alternate between 262 Hz & 310 Hz every 500 ms.
* Tri-colour LED will remain off.
* These **two** events will occur simultaneously, lasting 3500 ms.

1. *State of Emergency 2*

* LED blinks on and off every 100 ms.
* Speaker will alternate between 2000 Hz & 3000 Hz every 100 ms.
* Tri-colour LED will flash a random colour every 50 ms.
* These **three** events will occur simultaneously, lasting 5000 ms.

These parameters can easily be modified if one wishes to change the behavior of one or more alarms, including but not limited to the duration of the LED flash, frequency of the speaker tone being played etc. Refer to the Appendix for the full Arduino sketch used for the program, and details on how to modify these alarm parameters.

Results & Discussion:

Experiment CS01 :Evaluating the better method of alarm trigger

Experiments were conducted to determine the better method for alarm trigger. **Method 1** is for detecting if the RC time constant exceeds a certain, pre-defined threshold value and **Method 2** is for detecting if the increase in the RC time constant is equal to or greater than 15 within a 5 second period. As seen from the graph in Fig. 4, the *SCAR* analysis validated that **Method 2** excelled in A*daptability & Reliability*, and even surpassed **Method 1**'s *Consistency* score of 7.3 with an impressive 9. As a trade-off, it is slightly less sensitive than its former counterpart, 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 27% false alarm rate – more than 1 in 4 alarms could not be trusted.

This makes the verdict extremely clear – **Method 2** is 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.

Figure 4. *SCAR* analysis on two different trigger methods

Experiment CS01 Discussion

Initially in **Method 1**, even the slightest increases caused by an approaching hand would result in the exceeding of the pre-defined 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 security system as it caused the system to malfunction spontaneously from time to time (repeated sounding of false alarms). It necessitated a modification of the threshold value for the sensor to be operational again. The same issue applied to the *Adaptability* tests, which made the mean RC time constant change drastically. 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 CS02.

Experiment CS02 :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. 5.

Figure 5. *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.

A larger DTV would mean that more charges have to be drawn away from the sensor by the hand, resulting in more time required and/or a shorter distance needed between the hand and the sensor for the DTV to be met. 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. (Refer to the section Methods and Materialsfor passing criteria)

* *Consistency & Reliability* 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. 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. 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. 5, 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 CS02 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. 5. 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 security system that can safeguard an object against unwanted burglaries or theft. The combination of both sensors makes for a highly reliable 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. 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 *Sensitivity, Consistency, Adaptability* and *Reliability (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, finding the optimal resistor value for use in the capacitive sensor circuit (to possibly augment the *SCAR* of the sensor), 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/ .

[5] Chumbaka Asia. "*C520A Embedded System Guidebook for Arduino*". Retrieved 2015.

[6] Patricia, M. Feutech Engineers. (2008). "*Security Systems and Alarms*". Retrieved from http://www.feutechengineers.com/Security-Alarm.php .

[7] York College. The York College Physics II Laboratory Manual, "*The Time Constant of an RC Circuit*". Retrieved from https://www.york.cuny.edu/academics/departments/earth-and-physical-sciences/physics-lab-manuals/physics-ii/time-constant-of-an-rc-circuit

[8] STMicroelectronics Singapore (2009). Application note AN2927, "*RC acquisition principle for touch sensing applications*". Retrieved from http://www.st.com/web/en/resource/technical/document/application\_note/CD00222886.pdf

APPENDIX | 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;**  **elapsedMillis timerLED;**  **#define intervalLED1 500**  Frequency of LED blinking in *State of Emergency 1 & 2* is defined as 500 ms and 100 ms respectively.  **#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  Defines LCD display as 16 columns x 2 rows  **// ~~ LCD Set-up ~~**  **lcd.begin(16,2);**  Text printed under "Default" condition. This can be modified to print user-defined phrases.  **lcd.print("Object secure");**    **// ~~ 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:**  **tone (SpeakerPin,262,250);**  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.  **break;**  **case 1:**  **tone (SpeakerPin,310,250);**  **break;**  **}**  **if (timerMUSIC >= intervalMUSIC1)** { // after 500 ms…  **timerMUSIC -= intervalMUSIC1;**  // …reset the timer  **note = !note; }**  // …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.  **} // CLOSING "WHILE" FUNCTION**    Text printed under "Post-*SoE 1*" condition. This can be modified to print user-defined phrases.  **lcd.clear();**  **lcd.home();**  **lcd.print("Intruder alarm");**  **lcd.setCursor(0,1);**  **lcd.print("was sounded.");**  **digitalWrite(LEDPin, LOW);**  Hard-coded reset to ensure the LED will be off after the alarm protocol is finished.  **} // 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;**  **timerMUSIC = 0;**  **timerAlarm = 0;**  **timerAlarm2 = 0;**  **timerTriLED = 0;**  **while (timerAlarm2 <= 5000) {**    **// When object moved | LCD**  **lcd.home();**  Text printed under "S*oE 2*" condition. This can be modified to print user-defined phrases.  **lcd.print("WARNING!!! ");**  **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);**  **brightG=random(2);**  **brightB=random(2);**  **if (timerTriLED >= intervalTriLED)**  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.  **{**  **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();**  Text printed under "Post-*State of Emergency 2*" condition. This can be modified to print user-defined phrases.  **lcd.home();**  **lcd.print("WARNING: OBJECT");**  **lcd.setCursor(0,1);**  **lcd.print("WAS STOLEN!!");**  Move **currentState** to **previousState** to prepare for the next cycle.  **} // CLOSING "IF" FUNCTION**  **previousState = currentState;**  **delay(50);** // 50 ms buffer period  **} // END LOOP** |