

Final Report

Object Detection and Area Evaluation (ODAE)

Group 5 (Tuesday)

Mah, Eugene

Prasad, Vishant

Chen, Wei-Ming

Vora, Manan

MTRX2700 Major Project

Student Plagiarism: Course Work - Policy and Procedure


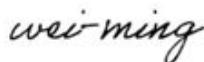
MTRX2700 Compliance Statement Individual/Collaborative Work

By submitting an assignment through the University Learning Management System (LMS),

I/We certify that:


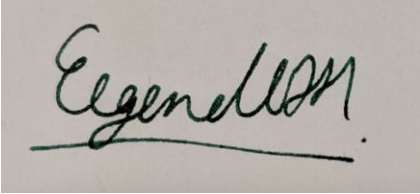

1. I have read and understood the University of Sydney Academic Dishonesty and Plagiarism Policy;
2. I understand that failure to comply with the above can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);
3. This Work is substantially my own, and to the extent that any part of this Work is not my own, I have indicated that it is not my own by acknowledging the source of that part or those parts of the Work.
4. I declare that this assignment is original and has not been submitted for assessment elsewhere, and acknowledge that the assessor of this assignment may, for the purpose of assessing this assignment: a) Reproduce this assignment and provide a copy to another member of Faculty; and/or b) Communicate a copy of this assignment to a plagiarism checking service (which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking).

Information on plagiarism is available online at: [University of Sydney policy website](#)

NAMES	SIGNATURES	SID	DATE
MANAN VORA		470318966	31/5/19
VISHANT PRASAD		470416309	31/5/19
WEI MING		470351332	31/5/19
EUGENE MAH		470177329	31/5/19

Group Certification

We certify that the design, implementation, and documentation presented by our Group are, unless otherwise acknowledged, the work of our group members only, and that the individuals named below were responsible for completing the following percentages of the modules named.

Name	%	Module	Signature
Manan Vora/470318966	40	Dragon 12	
	30	CodeWarrior	
	25	Report	
	25	MATLAB	
Vishant Prasad/470416309	20	Dragon 12	
	20	CodeWarrior	
	25	Report	
	15	MATLAB	
Eugene Mah/	20	Dragon 12	
	20	CodeWarrior	
	25	Report	
	15	MATLAB	
Wei Ming/470351332	20	Dragon 12	
	20	CodeWarrior	
	25	Report	
	45	MATLAB	

Student Plagiarism: Course Work - Policy and Procedure	2
Group Certification	3
1. Introduction	5
1.1 Document Identification	5
1.2 System Overview	5
1.3 Document Overview	5
1.4 Reference Documents	5
1.4.1 Acronyms and Abbreviations	6
2. System Description	6
2.1 Introduction	6
2.1.1 Inputs:	6
2.1.2 Outputs:	6
2.2 Operational Scenarios	7
2.3 System Requirements	7
2.4 Module Requirements: Lidar	7
2.5 Conceptual Design: Lidar Module	9
2.6 Functional Requirements: Seven Segment LED Module	11
2.7 Conceptual Design: Seven Segment LEDs	12
2.8 Functional Requirements: Gyroscope	14
2.9 Functional Requirements: Magnetometer	14
2.10 Functional Requirements: Acceleration	14
2.11 Functional Description: Servo Module	15
2.12 Conceptual Design: Servo Module	16
2.13 Conceptual Design: MATLAB GUI	18
3. User Interface Design	19
3.1 Interface Design	19
3.1.1 On-Board Interface (7 segment Display, on-board LCD)	19
3.1.2 Off-Board Interface (MATLAB GUI)	20
3.2 Interface Design	20
3.2.1 User Input and Output	20
5. Software Design	21
5.1 Software Design Process and Plan	21
5.2 Software Quality Assurance	22
6. Software Performance	22
7. Conclusions	23
8. References	23
9. Appendix	23

1. Introduction

1.1 Document Identification

This document describes the design of an **Object Detection and Area Evaluation (ODAE) System**. This document is prepared by **Group 5** for assessment in MTRX2700 in 2019.

1.2 System Overview

The ODAE will detect and measure the area of a shape positioned between **0.9 –1.3 meters** from the microprocessor heading. The object will be entirely mapped by scanning the following range: **+/-50 degree azimuth** and **+/-25 degrees elevation**. It should also provide the distance to the closest point. The planar object will fit in a rectangular area smaller than **40cm by 40cm** and in the centre will be perpendicular to the scanner unit in one azimuth orientation within the scanning area. There will be two different objects. One of the objects will have a rectangular shape and no holes. The other object will have polygonal shape and potential holes.

The Dragon12 Plus 2 Board will control a Pan and Tilt Unit (PTU) to scan the object and output the data to the PC in real-time (online) or offline using the serial port in a correct format and calibration. The output data is to be processed through MATLAB for a final shape plot/map. The system will be user interface and configurable via the Dragon12 Board (using the keypad as input and the LCD and 7 segment display as outputs) or via the PC using MATLAB.

1.3 Document Overview

This document will highlight the specifications of the ODAE. Furthermore, the different modules, their roles and issues that the group faced in the production of the system will be reported in this document. The system requirements is a significant section that will be discussed in this paper with diagrams, charts, graphs and tables to display essential information that was utilized in the designing of the system. Finally, the remaining documents that will accompany this are the user manual and the code files for the programming of the different modules.

1.4 Reference Documents

The present document is prepared on the basis of the following reference documents, and should be read in conjunction with them:

1. *The Project User Manual.*
2. *The Program Code.*

1.4.1 Acronyms and Abbreviations

Acronym	Meaning
PTU	Pan and Tilt Unit
IMU	Inertial Measurement Unit
IIC	Bus Protocol
PWM	Pulse Width Modulation / Modulated
D12P2	DragonBoard 12 Plus 2

2. System Description

This section is intended to give a general overview of the basis for the ODAE System design, of its division into hardware and software modules, and of its development and implementation.

2.1 Introduction

The function of the ODAE is to perform a scan of an area that is specified by the program (calibrated range of the pitch and yaw angles). Furthermore, these angles are to work in conjunction with a selected resolution and will display a scan on MATLAB. This plot is aimed to be created via the utilization of a 3D point cloud on the PC in two settings, **offline mode** or **real-time mode**.

2.1.1 Inputs:

- A 10 DOF Inertial Measurement Unit consisting of:
 - A 3DOF Gyroscope
 - A 3DOF Accelerometer
 - A 3DOF Magnetometer
- Lidar Laser Range Sensor v2

2.1.2 Outputs:

- Two Servo Motors
- Seven Segment LEDs
- Transfer Module
-

Inertial Measurement Unit (IMU)

Each measurement device within the IMU produces a 16 bit output on the x, y, and z axes. This is attached behind the Lidar Sensor. The combination of the 3 sensors used allows the

calculation of horizontal position (Azimuth) and vertical position (Elevation). These devices are communicated with using the I2C interface.

2.2 Operational Scenarios

The ODAE System is designed to scan at a distance between 0.8 and 1.2 metres. The system is aimed to be operated on-board. The system will be configured using the board's keypad with multiple buttons. The keypads are to output specific readings to the LCD. The circuit board requires the running of CodeWarrior. Next the PC requires connectivity to the board via a serial port (two USB connections required) where one will output to serial and one will power the board. The PTU will have a pre-programmed motion configuration. The program will further be aimed to allow for user configuration where the user may input variables that control the PTU using the keypad and measurements of pitch and yaw using the serial IO.

2.3 System Requirements

The operational scenario considered place certain requirements overall ODAE system, and on the modules that comprise it. The SCI1 modules are responsible for outputting and inputting values and parameters.

2.4 Module Requirements: Lidar

The Lidar Laser sensor comprises of a transmitter and receiver operating at 50Hz. This sensor outputs a PWM wave with a duty cycle directly proportional to distance at the rate of 1 msec/metre, with the maximum distance measured being 40 metres. Although it can be communicated with using the I2C interface, it does not work. Therefore, a trigger is used to begin the output wave for measurement.

Inputs:

- The Lidar Module takes a PWM wave input that has a duty time directly proportional distance. All Lidar functionality is controlled using two functions: an interrupt subroutine on timer 1 and the main calculation routine.

Process

- This uses an interrupt subroutine which toggles between high-going and low-going edge detection, and also saves the values into two time variables.
- The system first toggles the timer 1 input capture to detect a high-going edge on the PWM input. Then the routine waits for a second trigger to a low-going edge. After this all input capture is turned off to avoid any changes in values of time variables. A time difference is calculated, taking into account any overflow that may occur in the timer counter. This process is repeated 10 times and an average is found from the results.

Outputs:

- Information is output onto the seven segment LEDs in terms of centimetres
- Information is output to serial.

Timing

- The duty time directly corresponds to distance at the rate of 1msec/metre. Therefore, it is essential to be precise when detecting high and low going transitions on the input PWM.

Failure Modes

- The timer can be overflowed and an incorrect time difference and therefore distance can be output. However, this has been accounted for (see 2.6)

Performance

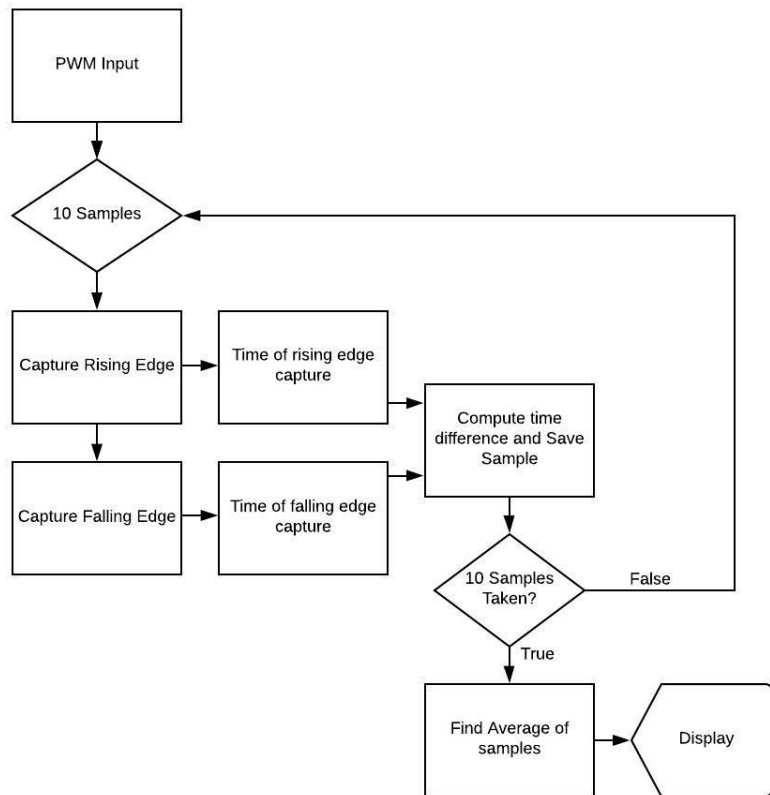
- Relatively constant performance time, this will change depending on the number of samples to be read by the module.

Design Constraints

- An ideal system would produce a fast output. However, this is limited by the effect of noise from the sensors, and it is essential to have a precise measurement.

2.5 Conceptual Design: Lidar Module

The following shows a flowchart for the Lidar Module



When calculating the time difference, it is important to take timer overflow into account. The DragonBoard 12's Timer Counter (TCNT) is a 16 bit register, overflowing to 0 when it reaches a full value of 65535. Therefore, if the falling time is lower than the rising time, the difference can be calculated using:

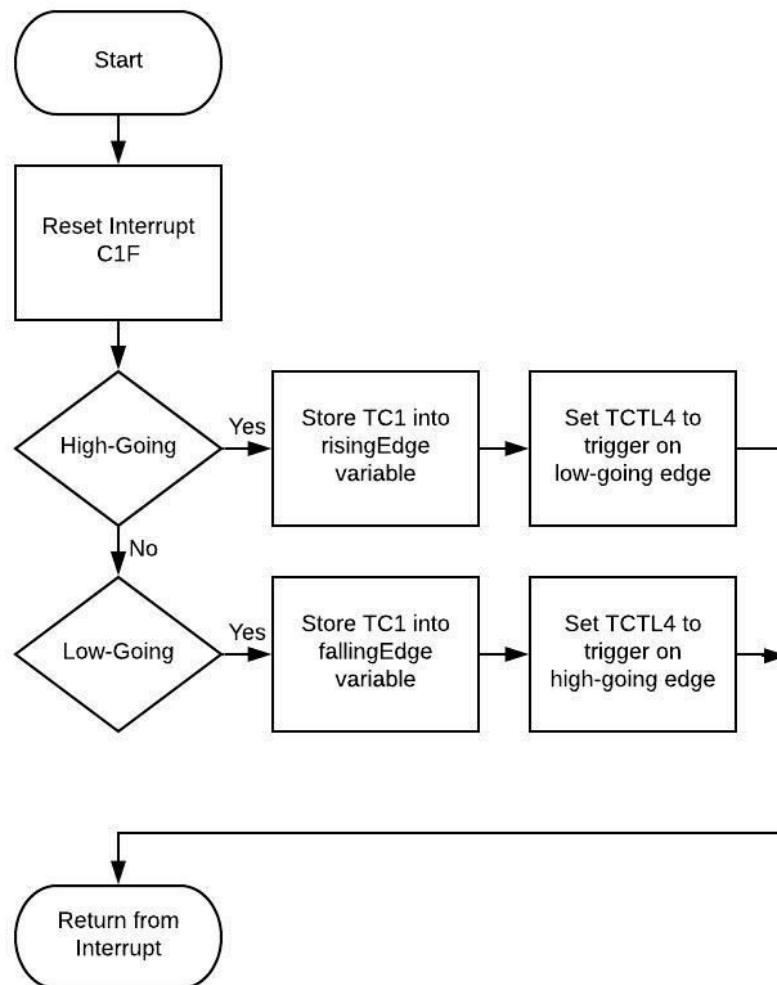
$$TimeDifference = fallingTime + (65535 - risingTime)$$

At every measurement, the time difference is a measure of time in terms of register values. This must be turned into values useful to the user, specifically distance. The DragonBoard 12's bus clock operates at a frequency of 24MHz. Therefore, the time difference must be converted to time by dividing the amount of clock cycles taken to reach 1ms, which is 24,000. In an equation format:

$$Distance(m) = Time(ms) = TimeDifference / 24000$$

Distance testing also showed the object was closer than measured, and thus an offset was added to account for this. This may have to be tested again depending on a range of distances required by the user.

All measurements are taken by interrupt generation on Timer 1. The following flowchart shows the control flow of the interrupt subroutine:



2.6 Functional Requirements: Seven Segment LED Module

This section covers the functional requirements to properly operate the Seven Segment LEDs.

Inputs:

- The Seven Segment LED Module takes a distance in centimetres as an input.

Process

- The LEDs must be selected by clearing the respective bit in the last four bytes of the PTP register. By splitting the input into its constituent parts, it is possible to match it up to a lookup table and get the order of bits required to light up the respective Seven Segment LED. After each selection/write process, there will be a small delay. This process is repeated for a couple of seconds before the system moves onto the next part.

Output

- Distance in centimetres displayed on the seven segment displays

Timing

- The timing for this module is in the moderate region as it requires the distance to be visible to the human eye.

Failure Modes

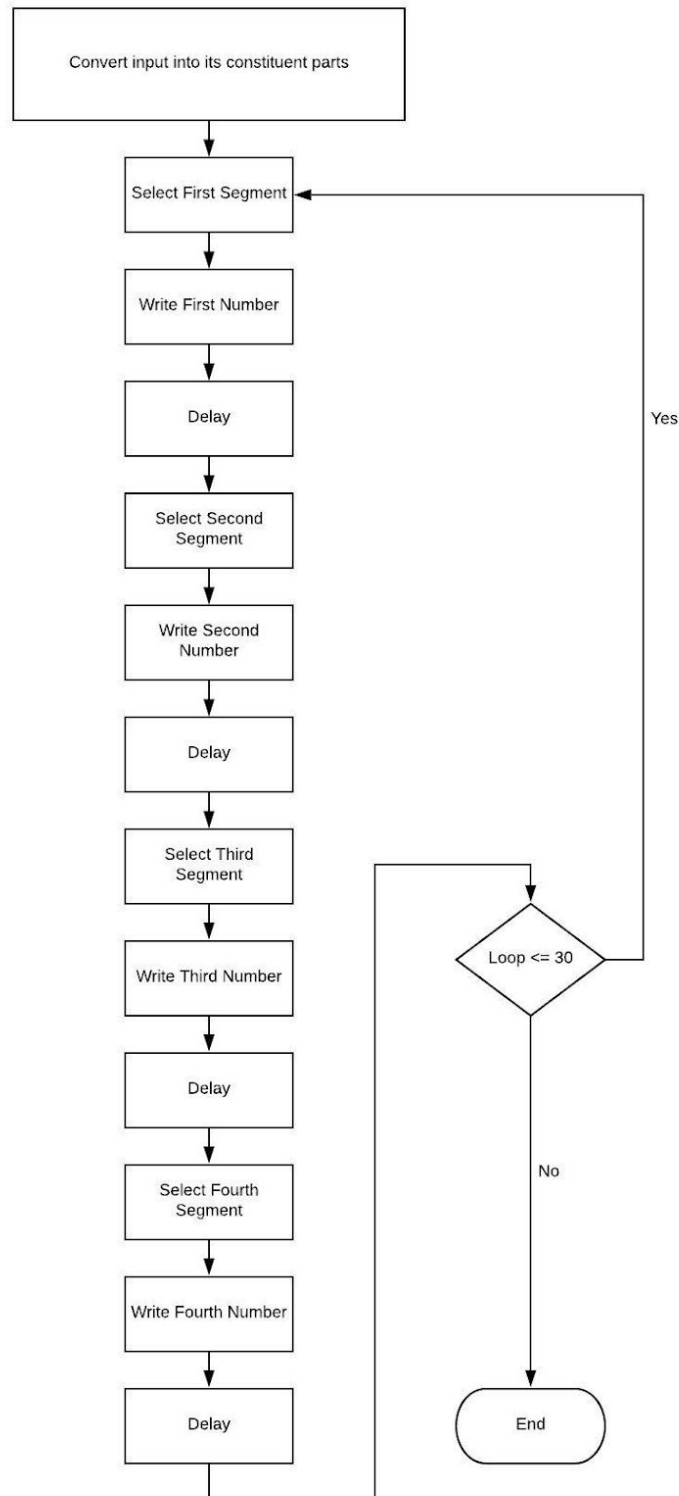
- This code only displays to the LEDs, does not account for failure modes.

Performance

- The delay used in this code does not affect any other interrupt subroutines.

2.7 Conceptual Design: Seven Segment LEDs

The flowchart below shows the control flow of the seven segment LEDs:



Breaking the numbers into their constituent parts involves producing four numbers from the input to display onto the four seven segment LEDs. This is essentially dividing by decreasing powers of 10 to find each number. The math is the following:

$$\begin{aligned}firstNumber &= (x - x\%1000) / 1000 \\secondNumber &= (x - firstNumber * 1000 - x\%100) / 100 \\thirdNumber &= (x - firstNumber * 1000 - secondNumber * 100 - x\%10) / 10 \\fourthNumber &= x\%10\end{aligned}$$

Where x is the input to the system.

The program contains two global lookup tables that allow:

1. The selection of seven segment LEDs
2. The selection of bit order required to display number at its index in the lookup table

This way, the program can efficiently lookup the pattern and select the seven segment LED required to display on.

2.8 Functional Requirements: Gyroscope

The gyroscope is a 3DOF sensor that outputs the rate of rotation based on the x, y and z axes.

Inputs

- Takes orientation of the unit based on how the unit is oriented in reference to a flat surface.

Process:

Outputs:

Timing

Performance:

2.9 Functional Requirements: Magnetometer

The magnetometer is a HMC5883L 3-Axis Digital Compass IC. It has been preloaded with a program to read and print out the raw data from the sensors onto the serial monitor.

Inputs:

- The orientation of the unit based on the Earth's magnetic field from the surroundings to the sensors

Process:

-

Outputs:

- The raw data of the sensors

Timing:

-

Performance

2.10 Functional Requirements: Acceleration

The accelerometer which is a ADXL345 model, has been preprogrammed to export raw readings from the sensors. The main goal is to interpret the raw data into readable values. In this case, the raw data from the accelerometer has been used to calculate the elevation of the unit.

Inputs:

- The acceleration of the entire unit when the unit is moving in the x,y & z axis.
- Gravitational force when the unit is stationary.

Process:

- Using the formula below, the x, y and z components are used to calculate the accelerometer at any tilt.
- $\phi = (A_{(Z,Out)} / \sqrt{A_{(X,Out)}^2 + A_{(Y,Out)}^2 + A_{(Z,Out)}^2})$

Output:

- The elevation of the unit in degrees.

Timing

- This module has to be run concurrently and therefore its efficiency or timing does not matter.

Performance

-

2.11 Functional Description: Servo Module

The PTU consists of two motors, connected to pins PP5 and PP7, which also act as output pins for the DragonBoard 12's PWM Module. The two servo motors are mounted in such a way that one provides horizontal movement (yaw) and the other controls the pitch of the Lidar. The position of these motors can be controlled by sending PWM waves with various duty cycles to the motors. This module operates at a frequency of 50Hz. Furthermore, this module is the integration umbrella of the whole system.

Inputs:

- Duty cycle to write to PWMx

Process

- The PWM signals required for these servos are first initialised before entering the main loop. This is done by setting certain registers (explained in section 2.15)
- PTU is moved to starting position at +25 degrees elevation and -50 azimuth.
- PTU is then moved horizontally until +50 azimuth. It then returns to the initial azimuth and lowers the elevation of the Lidar. This is then repeated until the elevation reaches -25.

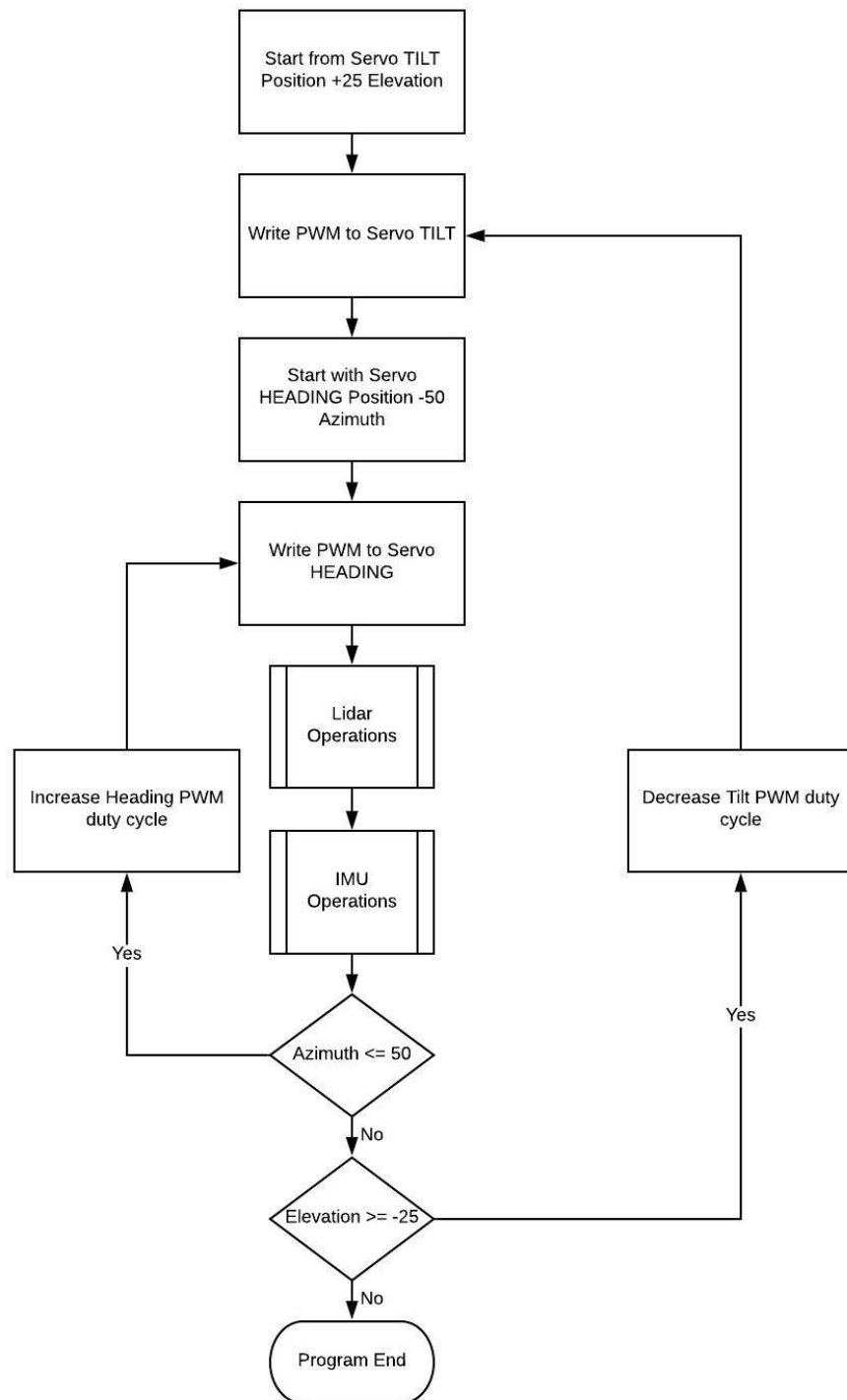
Output

- Servo moves to a new position incrementally in order to take measurements.

Timing

- No specific timing requirements

2.12 Conceptual Design: Servo Module



The PWM module allows the programmer to output PWM waves to devices, in this scenario, servo motors. The servo motors operate at a frequency of 50Hz, and thus a PWM wave of that frequency with varying duty cycles must be sent to the devices. However, since the bus clock is 24MHz, it must be scaled down to allow for a 50Hz frequency.

The PWM module has four clocks that may be used:

- Clock A and Clock SA are used to control PWM5
- Clock B and Clock SB are used to control PWM7

Clock A can be prescaled, and can be further scaled down to a value small enough to allow for smaller PWM signals, using clock SA. The same applies to Clocks B and SB.

Setting a period of 0.02 seconds requires setting a value in the PWMPERx register. From the PWM Module Documentation, the following formula is used to apply a value to PWMPERx:

$$\begin{aligned}
 PWMxPeriod &= ChannelClockPeriod * PWMPERx \\
 0.02 &= (1/24,000,000) * PWMPERx \\
 PWMPERx &= 480,000
 \end{aligned}$$

PWMPERx registers are 8 bit registers, that can hold a maximum value of 255. Therefore, clock scaling is required. Pre-scaling clock A down by 16 provides a 1.5Mhz frequency. At this frequency, the value required for PWMPERx is 30,000, which is still unwritable.

Therefore, clock SA must be used to scale clock A down further. The following formula provided by the document is used to find the scale value for a specified clock SA frequency, which should ideally be close to 50Hz.

$$\begin{aligned}
 Clock\ SA &= Clock\ A / (2 * PWMSCLA) \\
 PWMSCLA &= Clock\ A / (2 * Clock\ SA) \\
 PWMSCLA &= 1,500,000 / (2 * 50) = 15000
 \end{aligned}$$

As before, a value of 15,000 cannot be written to the 8 bit register of PWMSCLA. To bypass this, a different value must be used. 128 is the highest general pre-scaling value, so we will use 62 in order to pre-scale clock A further.

$$Clock\ SA = 1,500,000 / (2 * 62) = 12100$$

With Clock SA operating at 12.1 kHz, the PWMPERx value is calculated to be:

$$\begin{aligned}
 0.02 &= (1/12,100) * PWMPERx \\
 PWMPERx &= 242
 \end{aligned}$$

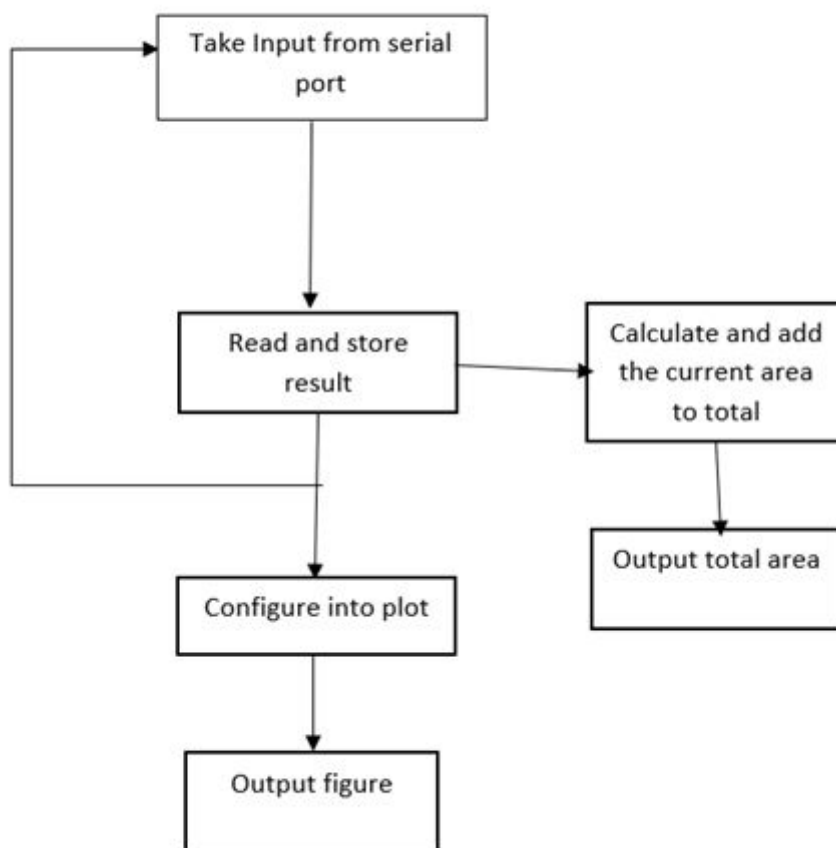
As Clock SA and SB are being used to obtain the 50Hz waves, they must be selected by setting the respective PWM bits in the PWMCLK register.

The respective PWM polarity bits (PWMPOL) need to be set, and output to be let aligned by clearing all bits in PWMCAE. This is to ensure the the duty cycle set occurs in the high state, and output is produced immediately.

Finally, the PWM channels need to be turned on by setting the respective bits in the PWME register. As a reference, the devices are first sent to their centre positions. This completes the initial setup. Throughout the program, the duty cycles of PWMs are changed according to the flowchart above. Please note that elevation and heading do NOT directly correspond to duty cycle values / servo positions.

2.13 Conceptual Design: MATLAB GUI

The MATLAB GUI comprises of 2 functions and 1 main program file. The program will take inputs from serial port. The main program will draw out the figure scanned while using one of the function to calculate the total area of the shape scanned. The other function is to set up the color interface of the figure draw by MATLAB.



Inputs:

- MATLAB GUI takes the input given out by lidar through serial port.

Process:

- Setting up the maximum x and y coordinate which the lidar can go, the maximum size of the figure can then be set using these maximum numbers
- Get the string from serial port and change it to numbers
- Check if the distance is over 1.4m since it only needs to detect things with this range
- If it is within the range, store the number into matrix and draw it

- The total area will be the area from the first row of matrix adds to the last row of matrix
- Store the value of the total area

Total area calculation:

Convert from degrees to radians first as matlab only uses radians. X is the row of the matrix while y is the column.

$x = 2 * \tan(50 * \pi / 180) * d;$

$y = 2 * \tan(25 * \pi / 180) * d;$

length x height is equal to the area of a single row in the matrix

$total = x * y$

Outputs:

- Information is output onto MATLAB interface
- Figure will be drawn

Failure Modes

- The distance from the lidar to the object cannot be too near or too far.
- Steps for the servos to move so to set up x and y maximum needs to be calculated accurately

Performance

- Figure was drawn and total area is outputting to the screen. However, it did not draw a perfect shape out

Design Constraints

- An ideal system would produce a more accurate figure and will be consistent.

3. User Interface Design

The user can choose to interact with the system with the off-board user interface GUI on the computer.

Upon initial set up (connecting the dragon 12 board to computer using USB cable. Connect the laser with ribbon cable and power it. Run the codewarrior program designed), the system will be at on-board configuration mode.

3.1 Interface Design

3.1.1 On-Board Interface (7 segment Display, on-board LCD)

Immediately after starting the program using codewarrior, the laser will start moving and scanning for objects in front of it. Measurements will be taken by the system.

Note: The GUI interface must be open with the correct COM configured before the system can start to measure.

The Seven Segment Display will display the output of the distance from currently inputted LIDAR measurement.

The LED display will output the current Azimuth and Elevation of the laser.

“Azimuth: -50 –+50”

“Elevation: -25 –+ 25”

Online plotting can be set within the GUI, real time points inputted from the system will be plotted in the GUI as 3D Point Cloud. It can also be plotted after measurements if real time is not required.

3.1.2 Off-Board Interface (MATLAB GUI)

The system be Off-Board Configuration mode after the MATLAB GUI has been opened. The COM port configured and the C button has been pressed on the board. System can be started by pressing “run” on MATLAB.

3.2 Interface Design

3.2.1 User Input and Output

With the ODAE GUI interface open:

When the program is running, servo will move automatically on its own, from left to right, up and down. The current status,(angle of the servos, accelerometer value distance detected by the lidar) will all be display onto PUTTY, user terminal. Meanwhile, current measured range will be displayed on 7-seg with 4 digit. A heading and the elevation of gyro/accelerometers/from servos will be displayed onto the LCD.

With MATLAB GUI interface open:

MATLAB GUI has to run with codewarrior in this situation. Startup and run MATLAB first before running codewarrior. As the system is running, the servos will start to move and the lidar will start to measure. Real time value measured will be sent to MATLAB via serial port. Instantly after receiving data from serial port, MATLAB will start plotting, outputting the shape of the object scanned. It will also output the total area of the object.

4. Hardware Design

All hardware are ready wired and constructed. The hardware involves a 5V power supply, 69-pin ribbon cable to the PTU, two 5V USB connection with a D12P2B. The PWM signal is a feature of the PTU and is utilized to control the servo motors as well as a LIDAR for range readings. The serial connection SCI1 and the IMU and LIDAR operate using IIC protocol.

5. Software Design

5.1 Software Design Process and Plan

The software for this module was designed from the lowest level functionalities to full integration. Initial Design and Unit testing was conducted on all measurement and movement modules, that is:

- Servos
- Lidar
- Accelerometer
- Gyroscope
- Magnetometer

- Along with block diagrams for each module (shown above)

When starting, an I2C interface and Serial Interface were provided to obtain inputs from the IMU. From there, functionalities were to be split up into their constituent parts. Meanwhile, Lidar and Servo modules and inputs were being tested.

Next, the output onto the seven segment LEDs and LCD were tested. It was found that to test IMU functionality properly, the servo's had to be programmed first.

Finally, a full integration took place. This involved calibrating all modules implemented and producing a MATLAB User Interface to display data and plot objects in real / offline time.

5.1.1 Software Development Environment

The following lists all software and hardware used in this system:

Software:

- Freescale CodeWarrior IDE
- PuTTY serial monitor
- MATLAB

Hardware:

- DragonBoard Plus 2 Development Board
- 2x HS-422 Servo Motors
- Lidar-Lite v2 Laser RangeFinder
- ADXL345 Digital Accelerometer
- L3G4200D UltraStable Three Axis Digital Output Gyroscope
- HMC5883L 3Axis Digital Compass IC (Magnetometer)

5.2 Software Quality Assurance

In order to ensure the program works as expected, all variables / units being tested had to be debugged by printing out values / checkpoints onto serial.

6. Software Performance

6.1 Performance Testing

6.1.1 Ease of Use

The ease of use of the system will be upheld by using a user manual and a description in the operational description. The LCD will also take steps to communicate to the user their input and configuration of the program. Furthermore, the serial output will have questions relating to the users configuration of the scan.

6.1.2 Accuracy

The system will output a 3D point cloud with data of the shape that the ODAE will scan. The accuracy of the scan would be limited by the configuration due to the angle calibration correlating with the PWM signals of the angles selected.

6.2 State of the System as Delivered

The servos are controlled by a PWM signal within the PTU from -25 to 25 degrees and -50 to 50 degrees in directions of yaw and pitch respectively.

6.3 Future Improvements

Future improvements to the system design and operation include:

- Displaying all necessary information to the LCD such as pitch and yaw.
- Calibrating the gyroscope and magnetometer.
- Configuring the keypads and LCD.
- Completing the serial output and user input configuration options.

7. Conclusions

The process involved countless adjustments to the approach of programming and finally the servo operation, LIDAR and accelerometer coding was well designed to specification and operated well however, the keypad and LCD was partially completed along with serial input/output. Finally, the plotting of the shape, gyroscope and magnetometer was incomplete due to significant problems in designing the program and operational misunderstandings.

8. References

Nebot, E. (2019). *MTRX2700 Mechatronics 2 Course Notes*. Sydney: The University of Sydney.

9. Appendix

MTRX2700
Major Project 2019
Object Detection and Area Evaluation
Doxygen Code

Authors:

Manan Vora

Vishant Prasad

Eugene Mah

Wei Ming

Fri May 31 2019

File Index

File List

Here is a list of all files with brief descriptions:

Major-Project-Files/main.c3
-----------------------------------	--------

File Documentation

Major-Project-Files/main.c File Reference

```
#include <hidef.h>
#include "derivative.h"
#include "iic.h"
#include "pll.h"
#include "sci1.h"
#include "io.h"
#include "l3g4200.h"
```

Macros

- #define **laser_wr** 0xc4
- #define **laser_rd** 0xc5
- #define **gyro_wr** 0xD2
- #define **gyro_rd** 0xD3
- #define **accel_wr** 0xA6
- #define **accel_rd** 0xA7
- #define **ADXL345_TO_READ** 6
- #define **ADXL345_POWER_CTL** 0x2D
- #define **ADXL345_DATAX0** 0x32
- #define **ADXL345_DATA_FORMAT** 0x31
- #define **ADXL345_OFSX** 0x1E
- #define **ADXL345_OFSY** 0x1F
- #define **ADXL345_OFSZ** 0x20
- #define **ALPHA** 0.5
- #define **magnet_wr** 0x3C
- #define **magnet_rd** 0x3D
- #define **HM5883_MODE_REG** 0x02
- #define **HM5883_DATAX0** 0x03
- #define **BUFF_SIZE** 100

Functions

- void **setAlarm1** (uint16_t msDelay1)
- void **delay1** (uint16_t msDelay1)
- void **Init_TC6** (void)
- void **adxl345_getrawdata** (int *axraw, int *ayraw, int *azraw)
- void **accel_init** (void)
- void **accel_calc** (void)
- void **hm5883_getrawdata** (int *mxraw, int *myraw, int *mzraw)
- void **magnet_init** (void)
- void **l3g4200d_getrawdata** (int *gxraw, int *gyraw, int *gzraw)
- void **gyro_init** (void)
- void **gyro_test** (void)
- void **Lidar_Init** (void)
- void **Init_TC1** (void)
- void **Lidar_Display** (float average)
- void **delay3** (void)
- void **Servo_Init** (void)
- void **Servo_Move_Pan** (int duty_cycle)
- void **Servo_Move_Tilt** (int duty_cycle)
- void **main** (void)

Variables

- volatile uint8_t **alarmSignaled1** = 0
 - volatile uint16_t **currentTime1** = 0
 - uint16_t **alarmTime1** = 0
 - volatile uint8_t **alarmSet1** = 0
 - int **k**
 - int **res1**
 - int **axraw** [BUFF_SIZE]
 - int **ayraw** [BUFF_SIZE]
 - int **azraw** [BUFF_SIZE]
 - float **pitch**
 - int **mxraw** [BUFF_SIZE]
 - int **myraw** [BUFF_SIZE]
 - int **mzraw** [BUFF_SIZE]
 - char **buff** [BUFF_SIZE]
 - int **gxraw** [BUFF_SIZE]
 - int **gyraw** [BUFF_SIZE]
 - int **gzraw** [BUFF_SIZE]
 - const unsigned char **segments** [4] = {0xFE,0xFD,0xFB,0xF7}
 - const unsigned char **display_numbers** [10]
 - int **initCount**
 - int **secondCount**
 - float **values** [10]
 - int **i**
 - int **x**
 - int **j**
 - int **m**
 - float **total**
 - int **numbers** [4]
 - unsigned short **diff**
 - float **average**
 - char * **buffer**
-

Macro Definition Documentation

#define accel_rd 0xA7

#define accel_wr 0xA6

#define ADXL345_DATA_FORMAT 0x31

#define ADXL345_DATAX0 0x32

#define ADXL345_OFSX 0x1E

#define ADXL345_OFSY 0x1F

#define ADXL345_OFSZ 0x20

#define ADXL345_POWER_CTL 0x2D

#define ADXL345_TO_READ 6

#define ALPHA 0.5

#define BUFF_SIZE 100

#define gyro_rd 0xD3

#define gyro_wr 0xD2

#define HM5883_DATAX0 0x03

#define HM5883_MODE_REG 0x02

#define laser_rd 0xc5

#define laser_wr 0xc4

#define magnet_rd 0x3D

#define magnet_wr 0x3C

Function Documentation

void accel_calc (void)

void accel_init (void)

void adxl345_getrawdata (int * *axraw*, int * *ayraw*, int * *azraw*)

void delay1 (uint16_t *msDelay1*)

void delay3 (void)

void gyro_init (void)

void gyro_test (void)

void hm5883_getrawdata (int * *mxraw*, int * *myraw*, int * *mzraw*)

void Init_TC1 (void)

void Init_TC6 (void)

void l3g4200d_getrawdata (int * *gxraw*, int * *gyraw*, int * *gzraw*)

void Lidar_Display (float *average*)

void Lidar_Init (void)

void magnet_init (void)

void main (void)

This function turns on the magnetometer

This function collects data from the magnetometer

Parameters:

<i>values</i>	<i>mxraw</i> , <i>myraw</i> , <i>mzraw</i> : pointers to arrays that will store raw magnetometer values
---------------	---

This function turns on the accelerometer

This function collects data from the magnetometer

Parameters:

<i>values</i>	<i>axraw</i> , <i>ayraw</i> , <i>azraw</i> : pointers to arrays that will store raw accelerometer values
---------------	--

This function tests for the presence of the gyro sensor

This function turns on the gyroscope

This function collects data from the magnetometer

Parameters:

<i>values</i>	<i>gxraw</i> , <i>gyraw</i> , <i>gzraw</i> : pointers to arrays that will store raw gyroscope values
---------------	--

A function that initialises two PWM signals to be written to PP7 and PP5

A function that moves the servos horizontally

Parameters:

<i>duty_cycle</i>	an integer value to be written to the PWM7 duty register
-------------------	--

A function that moves the Lidar vertically

Parameters:

<i>duty_cycle</i>	an integer value to be written to the PWM5 duty register
-------------------	--

A function that finds the average of 10 samples from the Lidar sensor and displays it onto the serial port

A function that displays the distance in centimetres onto seven segment LEDs

Parameters:

<i>y</i>	a float number that represents the distance variable to be output
----------	---

A Timer 1 Interrupt Routine that is used to find high and low going edges for Lidar sensor

A function that initialises Timer 1

A function that sets a boolean value for delay operations

Parameters:

<i>msDelay1</i>	a uint16_t value that represents the delay in milliseconds
-----------------	--

A delay function that runs for a specified time

Parameters:

<i>msec</i>	a uint16_t value that represents the delay in milliseconds
-------------	--

A function that interrupts every millisecond Used for delay

A function that initialises Timer 6 to be used in delay functions

void Servo_Init (void)

void Servo_Move_Pan (int *duty_cycle*)

void Servo_Move_Tilt (int *duty_cycle*)

void setAlarm1 (uint16_t *msDelay1*)

Variable Documentation

volatile uint8_t alarmSet1 = 0

volatile uint8_t alarmSignaled1 = 0

uint16_t alarmTime1 = 0

float average

int axraw[BUFF_SIZE]

int ayraw[BUFF_SIZE]

int azraw[BUFF_SIZE]

char buff[BUFF_SIZE]

char* buffer

volatile uint16_t currentTime1 = 0

unsigned short diff

const unsigned char display_numbers[10]

```
Initial value:= {  
    0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F  
}
```



```
int gxraw[BUFF_SIZE]

int gyraw[BUFF_SIZE]

int gzraw[BUFF_SIZE]

int i

int initCount

int j

int k

int m

int mxraw[BUFF_SIZE]

int myraw[BUFF_SIZE]

int mzraw[BUFF_SIZE]

int numbers[4]

float pitch

int res1

int secondCount

const unsigned char segments[4] = {0xFE,0xFD,0xFB,0xF7}

float total

float values[10]

int x
```

Index

INDEX