UNIVERSITY OF TORONTO

INSTITUTE FOR AEROSPACE STUDIES

4925 Dufferin Street, Toronto, Ontario, Canada, M3H 5T6

BARL Inspector

prepared by

Team 20 – Monday
Angus Fung (1001371231)
Ahmed N Amanullah (1001325773)
Ali Aftabjahani (1001666716)

prepared for

Prof. M.R. Emami

A technical report submitted for
AER201 – Engineering Design

TA: Tim Teng

April 8, 2016

1. Acknowledgements

We would like to thank Prof. Emami and especially our TA Mr. Teng for their guidance and helpful feedback throughout the semester. We would also like to extend our thanks to Collin Harry, the supervisor of the machine shop for his continual help, feedback and contribution of suggestions for the electromechanical design of our robot.

2. Abstract

The request for proposal (RFP) posted the need for an autonomous robot that investigates the status of heavy water containing barrels for a nuclear power plant. The aim of the project is to develop a functional proof-of-concept prototype that satisfies the overall objective as well as requirements posed by the RFP.

Our posed solution involves the usage of Infra-Red (IR) sensors in conjunction with a laser and a laser receiver to identify a barrel and its liquid level, while using a Ultrasound (US) sensor to lookout for the pipe obstacle. A Differential Drive with rigid Casters allow the robot to go completely straight and stay on track, while a 'pincer-shaped' arm allows the checking of both sides of the barrel. Actuation is provided to rotate the arm out of the way of the pipe by means of rotating a wheel against another. The entire robot is controlled by a Peripheral Interface Controller (PIC) microprocessor.

The following document is a compilation of the various phases involved in this project – planning and research, individual subsystem work completion and details on team integration.

Table of Contents

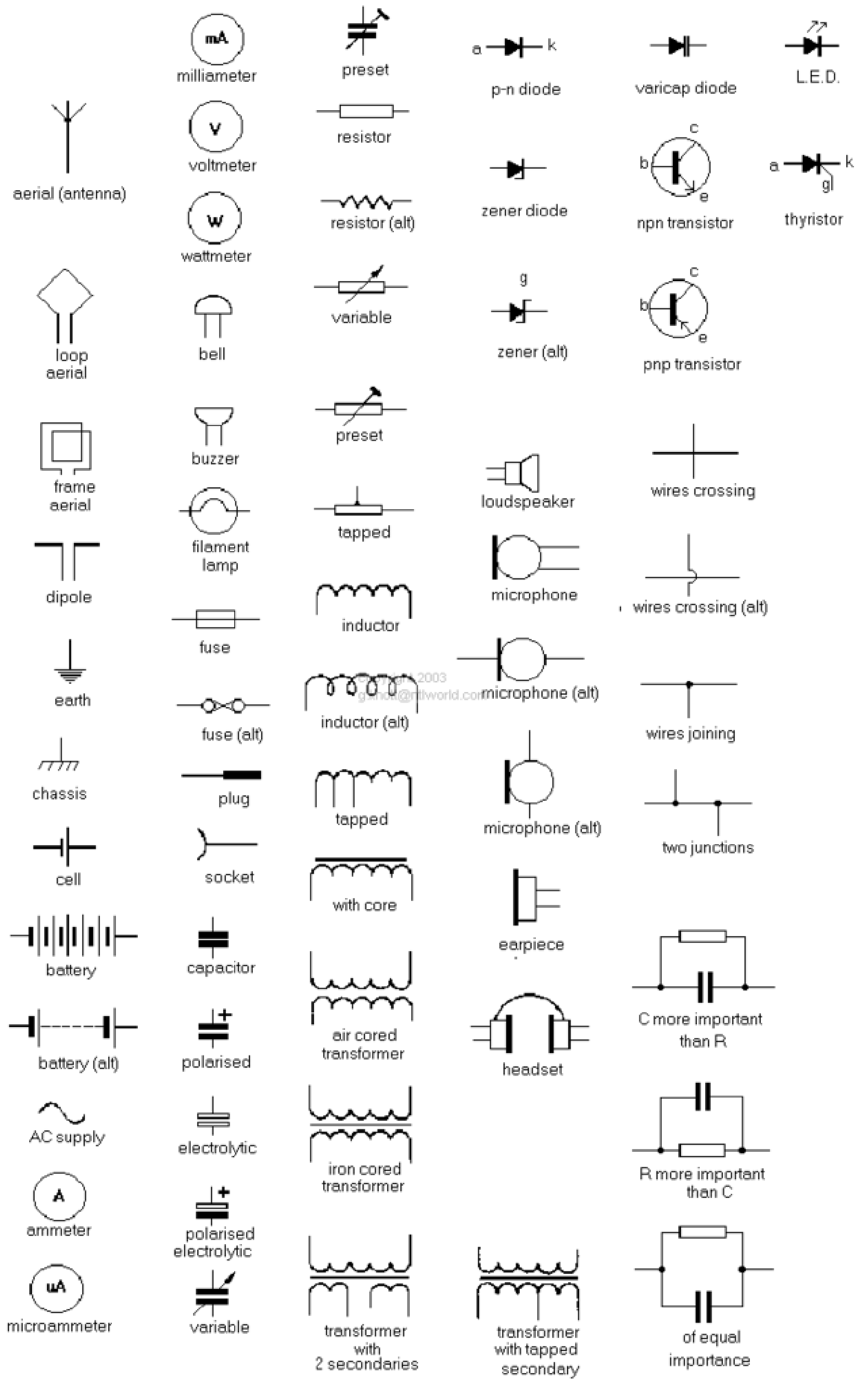
3. Symbols and Abbreviations

3.1 Abbreviations

Table 1: List of abbreviations

Acronym	Full form
RFP	Request for Proposal
IR	Infra-Red
US	Ultrasound
PIC	Peripheral Interface Controller
DFXs	Design for X’s
AHP	Analytical Hierarchy process
RTC	Real Time Clock
PCB	Printed Circuit Board
LCD	Liquid Crystal Display
DC	Direct Current
I/O	Input/ Output
PERT	Program Evaluation and Review Technique
ES	Earliest Start
LS	Latest Start
TF	Total Float
KISS	Keep It Super Simple
MISS	Make It Super Simple
SWAMI	Stored Waste Autonomous Mobile Inspector
RI	Relative Importance
RP	Relative Preference
CR	Consistency Ratio
FBD	Free body diagram
IC	Integrated Circuit
RAM	Random Access Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
ADC	Analog Digital Converter
PWM	Pulse Width Modulation
MUX	Multiplexer

3.2 Circuit Symbols



[5]

3.3 List of Symbols, units and constants

Table 2: List of symbols

Symbol	Description
t_0	Optimistic Time
t_m	Most Likely Time
t_p	Pessimistic Time
z	Standard Normal Variable
µ	Mean
σ	Standard Deviation
$\overline{x}$	Centre of mass x-co-ordinate (in metres (m))
Σ	Sum
M_y	Moment from y-axis (in Newton metres (N.m))
m	Mass (in kilograms (kg))
I_o	Polar moment of inertia (in kilogram metres squared (kg.m^2))
ρ	Density (in kilograms per metre cubed (kg/m^3))
dA	Differential Element of Area (metre squared (m^2))
θ	Angular Displacement (in radians (rad))
t	Time (in seconds (s))
ε	Element of
ω	Angular velocity (in radians per second (rad/s) or revolutions per minute (rpm))
α	Angular acceleration (in radians per seconds squared (rad/s^2)
τ	Torque (in N.m)
μ	Co-efficient of friction (dimensionless)
F_f	Friction force (in Newtons (N))
r	Radius (in m)
V	Volume (in metres cubed (m^3))
F_n	Normal Force (in N)
W	Weight (in N or pounds(lbs))
V	Voltage in voltage (V)

<u>Constants</u>

Pinewood ρ (old) = 500kg/m^3

μ = ~0.55 (coefficient of friction of tiling)

$Density\ of\ pinewood, \rho_{pw} = \sim 425\ kg/m^3$ [Toolbox]

$Density\ of\ hardwood, \rho_{hw} = \sim 340\ kg/m^3$ [Data sheet*]

$Density\ of\ plywood, \rho_{plw} = \sim 600 kg/m^3$

[http://www.plywood.cc/2009/03/27/plywood-density/]

$Density\ of\ Styrofoam, \rho_s = \sim 36.5\ kg/m^3$[Wikipedia – for now]

4. Introduction

In the modern day, as robot technology grow increasingly proficient and continue to replace humans in algorithmic, menial and particularly dangerous tasks. Robots dealing with toxic or nuclear waste for instance would eliminate the chance of risk of exposure from hiring otherwise human counterparts. That is the motivation for this project – to design an autonomous robot capable of checking heavy water inventory frequently for a nuclear power plant.

The goal of this project, however, is to design and manufacture a scaled down version of such a device, which is capable of travelling along a row of barrels and identify if the liquid level in the barrels are full, half-full or empty. Along the row it may meet obstacles – so there's also a need to implement manoeuvrability to avoid them. Since efficiency and cost and portability effectiveness are so important for pragmatic purposes, it follows naturally that the RFP prioritizes the related Design for X's (DFX's) and places operation time, size, weight and cost constraints. Our form of approach was to keep the design simple – so that it requires fewer components and expenses, while at the same time is feasible based on our skill set. In other words, we attempted to stay within the constraints while achieving the minimal functionality.

The project itself is divided into three sections – microcontroller (responsible for coding how the robot functions), electromechanical (responsible for building the system that is to carry out the functionality) and circuits (responsible for providing the interface between the control (microcontroller) to the body (electromechanical), as well as design the sensory input required to achieve objective) subsystems, with a team member assigned to each subsystem. Following the phase of us designing our subsystems, there is the phase of integrating all of our subsystems into the robot, and refining and debugging until we achieved the desired result.

The budget assigned is about \$230CDN and is funded via personal savings. By the end of the project our overall cost amounted to \$229.99CDN (so at the break-even point) and we managed to achieve partial functionality in which the individual components were functional but the overall robot was not. Needless to say the robot could not perform the run, so the accomplishment of the overall objective and satisfaction of the time constraint could not be ascertained.

5. Project Concept and Design Parameters

5.1 – Objectives and Constraints

Goal (High level objective): The BARL Inspector is designed to inspect and categorize barrels field with liquid to ‘full’, ‘half-full’ and ‘empty’.

(Detailed) Objectives:

1) Functionality:

(I) Has to be able to detect, as well as inspect barrels and classify them into categories of ‘Full’, ‘Half-Full’ and ‘Empty’.

Metric: Length measure of tape attached to barrel. If it is greater than 2/3 of height of barrel, should be classified ‘Full’; if it is more than 1/3 but less than 2/3 of the height of the barrel, should be classified ‘Half – Full’; otherwise should be classified ‘Empty’.

Constraint: Has to detect at least 3 barrels and attempt to identify their liquid level. Furthermore the robot has to stop at the end, and display this information in the termination message. This is the minimum for being qualified as a ‘run’.

(II) Has to be able to measure and record the location of the barrel centreline from the Start Line.

Metric: Distance in m/ cm.

Constraint: Measurement has to be within ±10cm of actual distance.

(III) When inspecting a barrel, the barrel must not be displaced or relocated or lifted or damaged in some way.

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied.

(IV) Has to be able to record the amount of tall and short barrels – i.e. has to be a way to classify barrel type. It should also be able to distinguish between the pipe and the barrels.

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied.

(V) Machine should be able to know the end of its run so that it can start returning. There should, thus, be a way for the machine to reverse its direction of travel (note that for locomotion it is only necessary to travel in a linear path back and forth).

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied.

(VI) Display the start message as well as the termination message along with the pertinent information (this includes operation time, amount of barrels of each type, the distance of the centreline of each barrel from Start Line, the categorization of each barrel and (optional) exact liquid level of each barrel).

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied.

(VII) Has to be able to measure operation time.

Metric: This is a Boolean condition – it is yes (1) or no (0).
Constraint: This is a Boolean condition – it must be satisfied.

2) Time Management/ Efficiency

(I) Has to be time-efficient in measurement of barrel's liquid level (i.e. tape height) and allow for less time required in between measurement of barrel heights.

Metric: Time (in s)

Constraint: (There is no strict upper bound yet for individual measurements but **entire inspection operation should not take longer than 3 minutes or 180 seconds**).

(II) Transitions of sensitive attachment to robot in order to avoid obstacles should take as less time as possible. The motive is to reduce time in this department in order to save some time for the inspection operation.

Metric: Time (in s)

Constraint: Same as in (I) for now.

3) Size and Weight/ Compactness and Portability

(I) An attempt should be made to make solution compact.

Metric: Volume (in cm^3)

Constraint: Prototype should completely fit within a 55X55X55 cm^3 envelope at all operation times. (Bonus: Completely fits in volume that is less than 25% of the maximum envelope volume allowed)

(II) An attempt should be present to make device as lightweight as possible.

Metric: Mass (in kg)

Constraint: Maximum allowed is 10kg. (Bonus: Prototype weighs no more than half of maximum limit)

4) Cost/ Affordability

(I) Attempt to be cost-efficient in design. Having less material/ components needed as well as using cheaper material/ components is a possible route.

Metric: in CDN$

Constraint: Budget is $230CDN

5) Safety/ Stability

(I) There must be an emergency STOP button that stops all mechanical moving parts immediately.

Metric: This a Boolean with the only values to be taken can either be yes or no (i.e. 1 or 0).

Constraint: This is a Boolean condition – it has to be satisfied.

(II) The design dimensions ought to be such that the machine remains stable during operation such that it does not tip over for instance (i.e. this design will certainly require a sensitive attachment to the robot in order to measure liquid levels – the design has to be such that the weight of the arm does not create sufficient moment to tip it over).

Metric: Torque or moment (in Nm or oz.in.)

Constraint: This is a Boolean condition – it must not tip over

(III) Some part of the operation may require the robot to use its sensitive attachment to scan the opposite side of the barrel. As such countermeasures should be in place to retract sensitive attachment when it meets obstacles – most notably, the pipe.

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied

(IV) (Bonus) Be capable of inspecting barrels not in a straight line. This objective may also be relevant in the sense that it is possible that the robot may go of track (for instance because of imperfect surface conditions) and some countermeasures may be needed to bring it on track.

Metric: This is a Boolean condition – it is yes (1) or no (0).

Constraint: This is a Boolean condition – it must be satisfied (Note that the bonus part is not strictly necessary)

(V) (Bonus) Machine operates quietly and smoothly with little or no noise or vibration.

Metric: The amount of noise can possibly be measured in decibels. Vibration is indirectly related to noise level – or can be measured qualitatively (such as the less the better).
Constraint: (No strict one enforced since it is a bonus objective)

6) Miscellaneous Bonuses

(I) Machine durably constructed and functions consistently under different indoor and outdoor conditions, including terrains.

(II) Little time/ effort necessary to set up and calibrate the machine in the field, and machine is modular such that parts can be replaced or repaired easily.

(III) Machine looks elegant

(IV) Machine can inspect barrels of different heights and diameters with little or no modification.

(V) Real date/time of each inspection displayed on LCD on standby mode.

(VI) Machine stores log information in permanent (EEPROM) memory.

(VII) The operation information can be readily downloaded on a PC.

(VIII) Remote controller can be used/ is present for remote operation (for starting and stopping, specifically).

(IX) Machine can measure exact liquid level of barrel.

5.3 – Acceptance Criteria in Decision Making

A weight is assigned to each of the above six requirements on the basis of how crucial the objective is towards the fulfillment of the overall objectives (this include factoring in what we stand to lose from ignoring this objective, how many constraints are enclosed within the objective to name a few criteria to arbitrarily assign the weights). The weights were then used in Utility Charts and Analytical Hierarchy Process (AHP) analysis to try to make some important decisions early on in the semester – decisions such as which Drive mechanism to use. Naturally the options that scored highest using this tools were held high in esteem, as the weights of the objectives used in such processes best reflect what we are designing for. However, we always had the final say in our decisions and did not give full liberty to the tools, which was mainly influenced by desire to keep the overall design simple and feasible as per our proficiency and skill sets.

6. Budget

Table 3: Budget

Item	**Qty**	**$/per**	**$**
Electromechanical			
1X10X4 Pinewood Shelving	2	4.80	4.80
Swivel (or Non-Swivel)	2	2.45	4.90
Lazy Susan	1	6.31	6.31
DC Motors-Wheels set	3	10.95	32.85
1-1/2 in. Corner Brace	1	1.34	1.34
2 in. Corner Brace	1	1.53	1.53
10 X 1 Flat Square-head	1	2.97	2.97

8 X 1-1/2 Flat Square-head	1	2.97	2.97
4-40X1 Rd. Soc. Mach Screw	2	2.97-50%	2.95
4-40 Hex mach Sc Nut Pltd	2	2.97	5.94
1/8 X 1 Rd Soc Stove Bolts	1	2.97	2.97
1-1/2 Wood Screws (flat squarehead)	1	0.57	0.57
1X48 Hardwood Rd. Dowel	1	4.73	4.73
Adhesive grit	1	8.17	8.17
Battery connex	1	1.75	1.75
			89.55
Microcontroller			
Real Time Clock(RTC) & Coin Battery	1	5.00	5.00
PIC and Dev Board	1	50.00	50.00
Printed Circuit Board (PCB)	1	15.00	15.00
Liquid Crystal Display (LCD)/Key	1	6.00	6.00
			76.00
Circuits			
IR emitter	8	0.75	6.00
IR receiver	8	0.63	5.04
US sensor	1	9.50	9.50
Laser beam	1	4.99	4.99
Laser receiver	1	4.50	4.50
Direct Current (DC) Adapter	1	3.87	3.87
Battery Holder	1	3.99	3.99
Batteries	1	3.00	3.00
Wires	1	7.75	7.75
Heat Sinks	1	2.80	2.80
PCB	1	5.00	5.00
Encoder	1	8.00	8.00
			64.44
Total			**229.99**

7. Division of the Problem

The development of the concept level functional prototype is divided into several phases. The first two months of the project involves dividing the work into three subsystems, with one member of a 3-man team responsible for one subsystem. Specifically the subsystems are referred to as the microcontroller, electromechanical and circuits subsystems. As elaborated on in the 'Introduction', the microcontroller member is responsible for coding how the robot functions (for instance, in our case, reading input from the IR sensors and Laser receiver circuit and processing the data to identify the liquid level as well as the barrel). Secondly, as mentioned earlier, the electromechanical member is responsible for designing the system, the shell if you will, that is supposed to perform the function (i.e. to be precise, the electromechanical member designs the system such that it can carry out the function it is supposed and the microcontroller member codes for). Finally the circuits member is responsible for designing the interface that allows the microprocessor to control the robot as desired as well as the circuitry that allows the robot to receive the required input signals and provide the desired output from it (an example is designing an H-bridge circuit that connects to the DC motor as well as the BUS leading into the PIC – the H-bridge circuit allows the microprocessor to control the direction of spin of the motor shaft as necessary).

Early on in the semester and prior to the proposal we distilled what was necessary to be done for our respective subsystem in order to allow ourselves to efficiently plan our design process. The division of the statement of work from that time are as follows:

7.1 – Electromechanical

1) Choose materials to be used, as well as motors and drive system. Go to a hardware store to start gathering resources needed.

2) Make back of the envelope calculations on feasibility of design based off cost/ mechanics/ kinematics calculations. (This connects to 1 – do 2 before 1)

i) (In collaboration with circuit member) test sensors and connect it to dimensional calculations.

3) Making the base of the robot.

4) Implement the rotatable shaft / gears needed to affect the rotation mechanism.

5) Fabricate the arm. (See 2 (i)) -> Sensor testing is crucial for the designing of the arm.

7.2 – Circuits

1) Sensors to acquire:
    a. Photoelectric (this will be to read the tapes)
    b. Ultrasonic (this will be to detect the pipe)
    c. Encoder (to keep track of distance)
2) Circuit design:
    a. Voltage/Current analysis
    b. Circuit layout
        i. Connection to microcontroller
        ii. Connection to driver board
        iii. Create a circuit for a motor
3) Calibration of the Input/ Output (I/O) devices
    a. Make sure sensors work
    b. Make sure motors drive
        i. DC motors must reverse polarity
4) Acquire Power supplies
    a. Voltage stabilizer for batteries
5) Circuit testing to prove functionality

7.3 – Microcontroller

1) Keypad and display interface (LCD)
    a. Initiates the robot

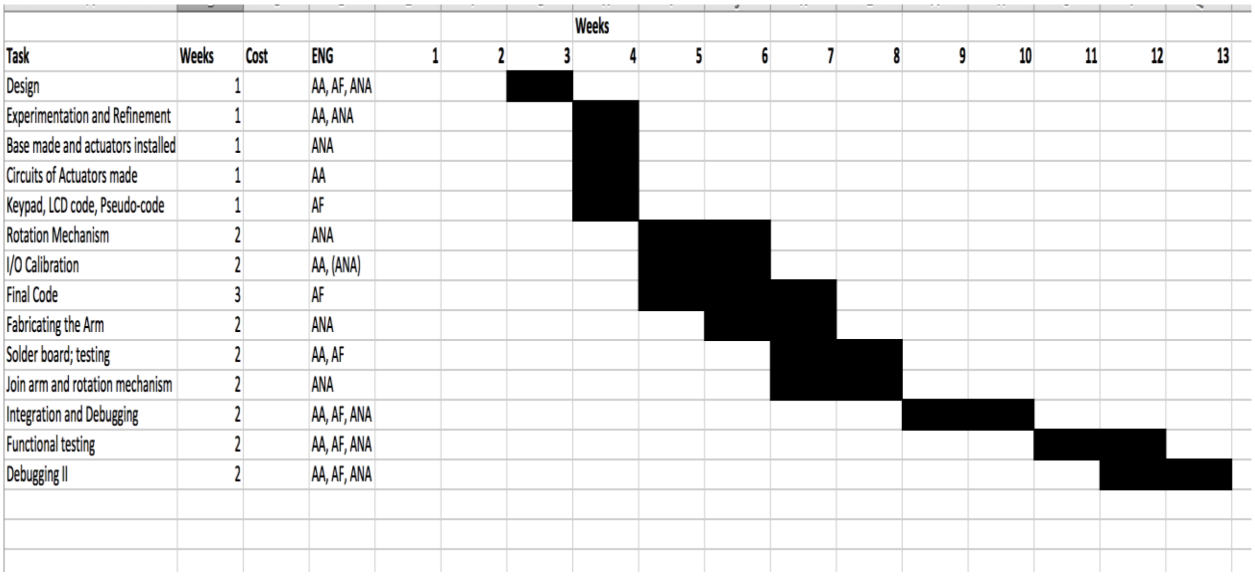
b. Display a completion or termination message on the LCD
c. Ready to communicate with the operator the inspection information
    i. Operation time
    ii. Total number of barrels for each type
    iii. Type of each detected barrel
    iv. Its location with reference to Start Line and level of liquid in it

2) Code for system debugging
3) Sensors:
    a. Photoelectric (this will be to read the tapes)
    b. Ultrasonic (this will be to detect the pipe)
    c. Encoder (to keep track of distance)
4) DC motor coding
5) Assist the Circuits member with fabrication and testing of solder boards

The division of the work statement allowed us to look at the problem one step at a time and design solutions for each aspect of the overall functionality separately – only to be integrated together later on to fulfill the overall functionality that is desired (hence we were designing for modularity). Afterwards we attempted to refine the final concept prototype as much as we could – but as said earlier we failed to accomplish the overall functionality of the robot.

7.4 – Initial and Accomplished Schedule (Gantt Charts)

As elucidated in the 'Division of the Problem' section, the initial phase of this project involved division of the overall problem into the subsystems and we drafted up our respective work statements to allow for efficient planning of our work throughout the semester. The tool used to graphically represent a distillation of our work statement is referred to as a Gantt chart and is as follows:



Task	Weeks	Cost	ENG
Design	1		AA, AF, ANA
Experimentation and Refinement	1		AA, ANA
Base made and actuators installed	1		ANA
Circuits of Actuators made	1		AA
Keypad, LCD code, Pseudo-code	1		AF
Rotation Mechanism	2		ANA
I/O Calibration	2		AA, (ANA)
Final Code	3		AF
Fabricating the Arm	2		ANA
Solder board; testing	2		AA, AF
Join arm and rotation mechanism	2		ANA
Integration and Debugging	2		AA, AF, ANA
Functional testing	2		AA, AF, ANA
Debugging II	2		AA, AF, ANA

Figure 1 Gantt Chart Prior to Proposal

Based on this Gantt Chart we drafted up prior to proposal we implemented Program Evaluation and Review Technique (PERT) analysis to try to confirm if our division of work allowed for finishing the project on time. The PERT analysis performed as of that time is as follows:

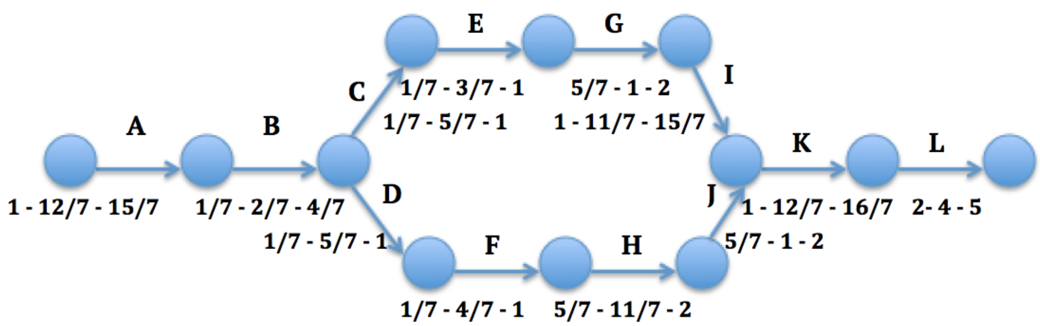


Fig. PERT analysis

KEY:
A: Design and Calculations
B: Material Acquisition
C: Base construction and base actuator instalment; Keypad, LCD and pseudo-code
D: Base construction; Keypad, LCD and pseudo-code; Circuit Analysis/Actuator and Motor circuits
E: Circuit Analysis/Actuator and Motor circuits; Finish up code started in activity **C**
F: Installing wheels and actuators for locomotion; Experimentation and Refinement
G: Experimentation and Refinement; I/O calibration; Starting final coding.
H: I/O Calibration; Rotational Mechanism; Final Coding
I: Implement Rotational Shaft Mechanism/ Start Arm Fabrication; Final coding
J: Arm fabrication; Final coding (cont'd)
K: Fabrication and Testing of Solder boards; Finish Arm/ Join Arm and Rotational Mechanism
L: Integration and Debugging.

Figure 2 PERT Chart

Table 4 PERT Analysis

Activity	t_0	t_m	t_p	t_e	ES	LS	TF	Variance
A	1.000	1.714	2.143	1.667	0.000	0.643	0.643	0.036
B	0.143	0.286	0.571	0.310	1.667	2.310	0.643	0.005
C	0.143	0.714	1.000	0.667	1.976	2.643	0.667	0.020
D	0.143	0.714	1.000	0.667	1.976	2.619	0.643	0.020
E	0.143	0.429	1.000	0.476	2.643	3.310	0.667	0.020
F	0.143	0.571	1.000	0.571	2.643	3.286	0.643	0.020
G	0.714	1.000	2.000	1.119	3.119	3.786	0.667	0.046
H	0.714	1.571	2.000	1.500	3.214	3.857	0.643	0.046
I	1.000	1.571	2.143	1.571	4.238	4.905	0.667	0.036
J	0.714	1.000	2.000	1.119	4.714	5.357	0.643	0.046
K	1.000	1.714	2.286	1.690	5.833	6.476	0.643	0.046
L	2.000	4.000	5.000	3.833	7.524	8.167	0.643	0.250

From the above PERT flow chart, the critical pathway was deduced, from which the following parameters were obtained: a T_e of ~11.357, and a σ_e^2 of ~0.470.

So, now let's see if we can finish project on time:

let's say we wish to calculate probability we finish on time,

P(t < 12 weeks)

$z = (t - \mu)/\sigma = 0.937748761$

So,

$P(z < 0.938) = P(z < 0.93) + 0.8*(P(z<0.94) - P(z<0.93)) = 0.825874$ [Used linear interpolation here]

<u>And we have an approximately 82.6% chance of completing project on time.</u>

To show our progress over the semester, the updated Gantt chart is as follows:

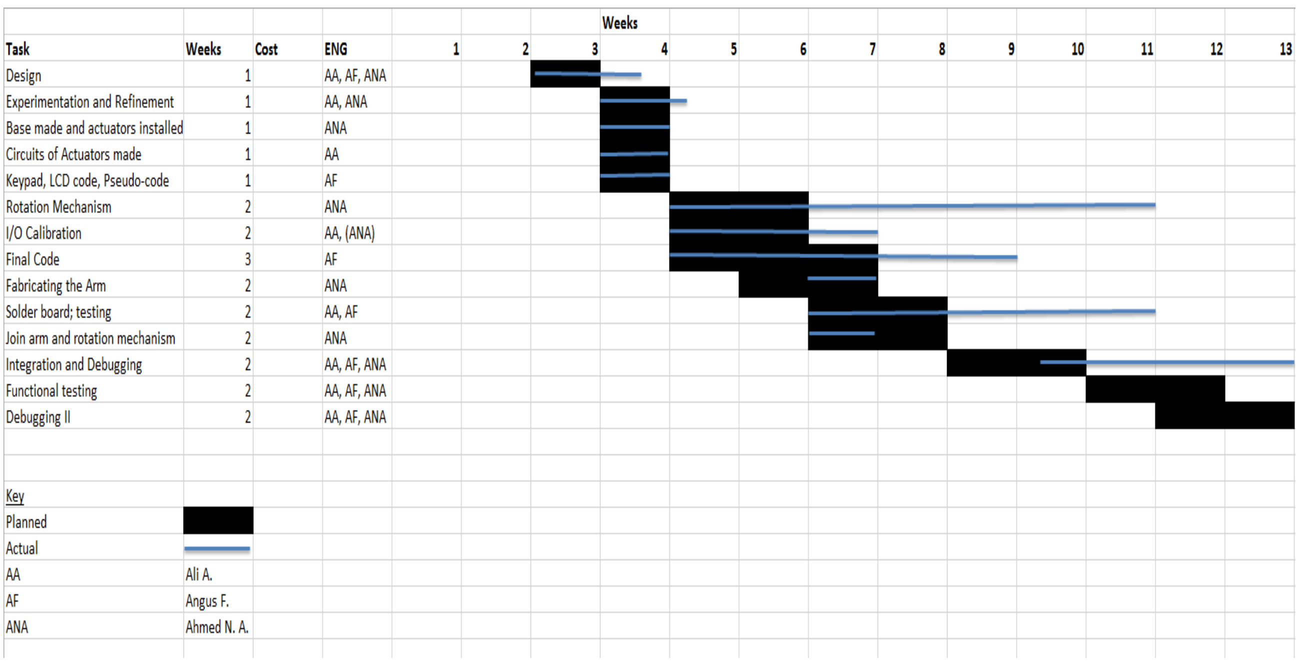


Figure 3 Gantt Chart at the end of the course

<u>Note:</u> 1) As mentioned in the 'Electromechanical Subsystem' section, owing to defects in the design of the rotation mechanism, the design of the actuation system that rotates the arm was an iterative process. I was only until the second team evaluation when the issues concerning the arm and actuation was rectified (full details of the iterative process is available in the electromechanical subsystem)

2) Owing to the delay experienced in debugging and completing our own respective subsystems, little time was left for the integration phase. We could only

manage to ensure functionality of the individual parts by the end, but we could never get to overall functional testing of the robot and its subsequent debugging phase.

8. Perspective

8.1 – Design Theory

The machine was designed for the various design elements: optimization, simplicity, modularity, durability, and safety. In particular, design iterations in the integration phase reflected these design values. Optimization was a key component in the software and hardware components of the machine. In the software, elegant algorithms as oppose to brute-force algorithms were utilized. During the integration phase for example, it was realized the operation optimization could be improved by changing the material of the robot arm. Furthermore, it was noticed that only one laser sensor was required, so the redundant sensors were removed to optimize performance.

The design concept of KISS and MISS (Keep it Super Simple and Make It Super Simple) was the key moto of our design (425). In every case where the designed seemed convoluted, brainstorming was performed as a team to find simpler solutions. Our machine consists of few moving parts and a simple design, which is essential as it reduces the number of problems that may arise. In a sense, this produces a more reliable machine that is less prone to mechanical problems. Should these problems arise however, debugging should take minimal time as the source of the problem could be quickly traced. The design decision of using an elastic band on the chassis was a simplistic design that accommodated for the imperfections of the circular arm base.

Modularity was also stressed in both the design and the design process. The division of the work statement allowed us to look at the problem one step at a time and design solutions for each aspect of the overall functionality separately – only to be integrated together later on to fulfill the overall functionality that is desired (hence we were designing for modularity). In the software, modularity was important as it allowed for the components to be tested individually and in conjunction with any other component.

The focus of durability is mostly in the design of the machine itself. The choice of material used was chosen to be sturdy so that the machine could operate consistency and expectedly each run.

Most importantly of all, safety was considered to ensure safety operation of the machine and the operator thereof, and an emergency stop was placed at the front of the machine to allow the machine to stop promptly when necessary should an emergency arise.

8.2 – Research Survey (History)

A similar project, as it relates to barrel waste inspection, was developed in 1993 by SWAMI, the Stored Waste Autonomous Mobile Inspector [1]. It was designed to tackle the issues of inspecting waste drums, and the following are the motivations of the project:

- Reduce personnel exposure to potential hazardous radiation when inspecting thousands of radioactive waste drums. Theretofore, all known waste storage facilities performed inspection manually – a serious health concern.
- Increasing cost effectiveness (operating cost is just the cost of operating the robot)
- Reducing the variation of inspection that would exist in manual inspection, as inspection thousands of waste drums is laborious.
- Eliminating the costs of hiring competent and experienced inspectors.

- The drums are located on *four* levels, and the accuracy of measurement of measuring the fourth level will be the same as that of the fist level. [1]

In many ways, this dilemma is similar to the problem with detecting radioactive hard water levels in barrels. Although the physical design of the SWAMI is not directly addressed in the paper, it seems that sensors are used to detect the locations of the drums and data is stored including geometric data (i.e height), location, and time stamp, which is not unlike our machine. A schematic or diagram of the SWAMI is shown below:

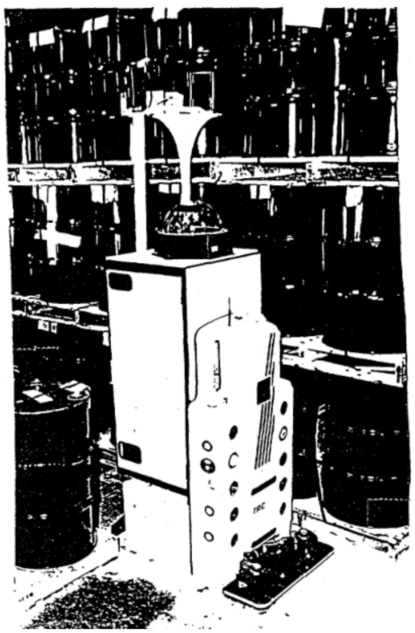


Figure 4: SWAMI

Due to the fact that SWAMI has to detect drums on four levels, a physical “arm”, which is a key component of our design, is not applicable. This reaffirms that radiation waste detection is a prevalent problem, and our proposal seeks to provide a solution for waste water detection in barrels.

8.3 – Limitations

The machine has limitations due to basic physics, the limitations imposed by the RFP, and the current state of technology. For the PIC microcontroller, limitations include the size of the look-up table, the amount of space in program memory, the time required to receive, process, and send signals, and the number of pins such as analog and PWM pins. For example, while PWM was very much necessary for the motor arm, due to there being only two PWM pins, only the motor wheels were allocated for PWM.

Since the robot is required to move along the aisle containing the barrels, the motor used to allow the robot to move should be able to provide the necessary torque. The motor we chose is capable of providing a torque of 1kgcm []. A quick sneak-peek to the 'Electromechanical Subsystem' section calculations would reveal that the motors are capable of moving a robot only if it weighs less than or equal to ~2.3kg (the physical equations used to obtain the result are disclosed in the same section in which the calculation was performed). This limits our choice of material and structural dimensions – the only way to allow for locomotion of a heavier robot would be to get more powerful motors, which are significantly more expensive and we would definitely go over budget.

Integration

9. Electromechanical subsystem

Requirements:

- ➔ Assessment of the problem (100)
- ➔ Solution (150)
- ➔ Supporting Calculations, Computer Programs, Simulation Results (100)
- ➔ Suggestions for Improvement (50)
- ➔ Tables (50)
- ➔ Figures (200)

9.1 – Assessment of the problem

The overall objective of the BARL Inspector is to travel 400cm and back, while on the way inspect a white measuring tape on several barrels (either on the side of the robot or against it) without disturbing the barrels or knocking into a pillar on the way. Furthermore, several other requirements are essential as well, which includes the constraints of run time not exceeding 180s, overall cost not exceeding $230CDN, total weight not exceeding 10kg and the necessity of an emergency stop button. The formulation of the solutions such that these requirements and overall objective are satisfied is broken and presented in the following subsections.

9.2 – Locomotion: Travelling 400cm and back

The choice of the drive system as well as the design of the base of the robot is crucial for the fulfillment of the overall objective. Previous to the submission of the proposal and during our preliminary design-brainstorming phase, we were debating between using a Differential Drive, an Omni Drive or a Track Drive for our drive system (insert images of each systems in figures? In this section?). For the choice of the system, we implemented the AHP technique to set the Relative Importances (RI's) and Relative Preferences (RP's) of our objectives and design solutions (i.e. the Drives) respectively. The utilization of this technique to check the Consistency Ratio (CR) for the computation of the RI's:

For the upcoming tables, key:

Objective 1 Functionality
Objective 2 Time efficiency
Objective 3 Size and Weight
Objective 4 Affordability
Objective 5 Safety/Stability
Objective 6 Bonuses

Table 5: The R.I. matrix of the objectives. The R.I.s are decided based on how crucial each objective is in terms of its impact on the overall goal as well as satisfaction of constraints

RI Factors	Objective 1	Objective 2	Objective 3	Objective 4	Objective 5	Objective 6
Objective 1	1	3	1	2	3	9
Objective 2	0.333333333	1	0.5	0.5	0.5	3
Objective 3	1	2	1	1	0.5	5
Objective 4	0.5	2	1	1	0.5	4

Objective 5	0.333333333	2	2	2	1	3
Objective 6	0.111111111	0.333333333	0.2	0.25	0.333333333	1

Table 6: Column normalized R.I. matrix

RI Factors	**Objective 1**	**Objective 2**	**Objective 3**	**Objective 4**	**Objective 5**	**Objective 6**	**Overall Importance**
Objective 1	0.305084746	0.290322581	0.175438596	0.296296296	0.514285714	0.36	0.323571322
Objective 2	0.101694915	0.096774194	0.087719298	0.074074074	0.085714286	0.12	0.094329461
Objective 3	0.305084746	0.193548387	0.175438596	0.148148148	0.085714286	0.2	0.184655694
Objective 4	0.152542373	0.193548387	0.175438596	0.148148148	0.085714286	0.16	0.152565298
Objective 5	0.101694915	0.193548387	0.350877193	0.296296296	0.171428571	0.12	0.205640894
Objective 6	0.033898305	0.032258065	0.035087719	0.037037037	0.057142857	0.04	0.039237331

Table 7: The P matrix. The eigenvalue of this matrix is calculated to give the consistency ratio (CR). The λ_{max} of this matrix was calculated using MATLAB to be 6.0, giving a CR of 0.

RI Factors	**Objective 1**	**Objective 2**	**Objective 3**	**Objective 4**	**Objective 5**	**Objective 6**
Objective 1	1	3.430225492	1.752295396	2.120871035	1.573477513	8.246517233
Objective 2	0.291526024	1	0.51083971	0.618289101	0.458709644	2.404074383
Objective 3	0.570680036	1.957561208	1	1.210338759	0.897952204	4.706122752
Objective 4	0.471504388	1.617366373	0.826214969	1	0.741901552	3.888269064
Objective 5	0.63553498	2.18002829	1.11364502	1.347887731	1	5.240950165
Objective 6	0.121263313	0.415960507	0.212489145	0.257183848	0.190805096	1

Note that the consistency check was also used for the 6 RP's generated using this technique, but is omitted for the sake of brevity. The end result, though, of using the RI's and RP's in establishing the decision is as follows:

Table 8: Design Decision of Choosing a Drive System. Note that (A), (B) and (C) represents Differential Drive, Omni Drive and Track Drive respectively.

	Objective	1	2	3	4	5	6	
	Overall Importance	0.323571322	0.094329461	0.184655694	0.152565298	0.205640894	0.039237331	**Total**
Preferences	**A**	0.333333333	0.333333333	0.5	0.5	0.538961039	0.333333333	0.431822297
	B	0.333333333	0.333333333	0.25	0.25	0.163780664	0.333333333	0.270364621
	C	0.333333333	0.333333333	0.25	0.25	0.297258297	0.333333333	0.297813081

Based on the AHP decision-making tool utilized, our choice is the Differential Drive system (which is in fact, the decision we came to). However it must be emphasized that our main reason for selecting the Differential Drive was for the sake of affordability (it requires the least amount of raw materials and expenses) and simplicity, and the AHP tool only served as reinforcement to the decision we made.

After the choice of the drive system, it was imperative to decide whether to account for adjustment of the vehicle direction during operation in the off chance that it deviates from the path. This is an important factor to account for as the slightest deviation from the path can lead to upsetting the barrels or colliding with the pipe-obstacle if not accounted for. Originally the team consensus was to have two rigid castor wheels at the front while using two wheels at the back rotated by a DC motor each – our reasoning being if we supply the same voltage to each motor and we orient the robot correctly prior to the run, it will continue to travel straight and the possibility of a deviation will not arise. However, experimentally running the base led to the observation that owing to imperfections in the flooring, the robot does go off track and the original rigid front wheels did not account for rotation. Hence swivel caster wheels, to allow for adjustment of direction during runtime, then replaced the front wheels. However, further experimentation (on the day of the demo no less) revealed that owing to the swivel caster wheels, the robot deviates too easily. So a recommendation for improvement would be to just stick with the rigid castor wheels, but to refine the design to allow for equal torque delivery to each motor so that the robot keeps going straight.

So, all in all, the final consensus is the design of a locomotion that travels 400cm in a straight line and returns (by reversing the polarity of the DC motors by means of an H-bridge to make the robot travel in reverse)

9.3 – Sensors mount design

While the base design and Drive system is crucial for the fulfillment of the overall objective, the fulfillment of its role as a BARL inspector is accomplished with the aid of sensor input. Hence it is necessary to design a satisfactory mount or attachment to the robot on which the sensors are strategically placed such that the robot can fulfill its overall objective. To elaborate, the white tape (against the black barrels) that is used as measure of the 'liquid status' of the barrel is located on side facing towards or away from the robot. In other words, it is necessary for the sensor to be in close proximity to each side – so a mount capable of allowing the sensor to be in such a position. At the same time care must be taken to not upset the barrel or bump into a column that is located in the same straight line in which the barrels are placed. About four versions of such a mobile mount (or 'arm' to be precise) were brainstormed in the initial planning (the images of each design is available for seeing in the "figures" section). In the end, we decided to go with the "Sensitive Arm Attachment" design, for the same reasons because of which the Differential Drive was chosen – on the basis of feasibility, affordability and simplicity. AHP technique was utilized for making this decision as well, the conclusions of which are as follows:

Table 9: The conclusive results are given below. Objectives 1, 2, 3, 4, 5 and 6 are Functionality, Time efficiency, Size and Weight, Affordability, Safety/Stability, and Bonuses respectively. 'A' represents the chosen design, while B, C, and D are the alternatives (see "figures" section for visual image).

		Preferences			
Objective	**Overall Importance**	**A**	**B**	**C**	**D**
1	0.323571322	0.25	0.25	0.25	0.25
2	0.094329461	0.571428571	0.142857143	0.142857143	0.142857143
3	0.184655694	0.45467033	0.14114011	0.14114011	0.263049451
4	0.152565298	0.465819398	0.0959699	0.161070234	0.277140468
5	0.205640894	0.25	0.25	0.25	0.25
6	0.039237331	0.142857143	0.285714286	0.285714286	0.285714286
	Total	0.346836277	0.197693358	0.20762541	0.247844954

9.4 – The evolution of the 'arm' and its actuation

As highlighted in the earlier sub-section, the purpose of the BARL Inspector is to investigate the liquid level of each barrel indicated by a strip of white tape (against the black barrel) located in the side of the barrel facing the robot or away from it. Hence the "Sensitive Arm" attachment is designed in a pincer-like shape (see 'figures' section for visual representation) with each pincer jaw to act as a mount for the sensor circuits – that way the sensors can inspect a white tape against black regardless of if it is on the side facing the robot or against it. However this creates another issue to be dealt with – the pipe obstacle (which is placed in an indefinite position along the linear path of barrels) is obviously taller than the robot (due to size constraints) and to avoid it as requirements specify the arm must be retracted in some way when facing the pipe. Our solution for retracting the arm was to rotate it by 90 degrees clockwise when facing the pipe and then rotating it back to original position after passing the pipe (the position of the pipe can be detected using a US sensor). When returning to the start line after travelling 400cm, the arm would stay in its retracted position to avoid double counting/investigating the barrels.

As to the details of the arm actuation, the design underwent an iterative development throughout the second half of the semester owing to imperfections in the actuation performance of each iteration. The very first design involved the connection of the 'pincer-shaped' arm (which by the way is constructed using pinewood) to a custom-made pinewood wheel (sometime later in the iterative process, was replaced by a plywood wheel because original piece was faulty) by means of a hardwood dowel (for visual representation and details see 'figures'). The pinewood wheel rests on top of a Lazy Susan, which in turn is mounted on a square piece of pinewood (the part as fashioned out to have the exact dimensions as the Lazy Susan). The concept is that if the Lazy Susan turns so does the dowel connected to the wheel resting on the Lazy Susan, which ultimately fulfills the function of rotating the arm. The actuation required to move the arm is accomplished by mounting a DC motor such that the wheel connected to the DC motor is in tight contact with the pinewood wheel. Thus by providing power to the DC motor, the shaft rotates the wheel, which in turn rotates the arm by contact friction between the DC motor wheel and pinewood wheel.

The issue with the first design was two part – firstly, the pine wood wheel had a smooth curved surface which resulted in frequent slipping in the contact between the wheels, and secondly (and more importantly) owing to imperfections (which result from carving the wheel using a band saw, so human error create an imperfect circle) in the pinewood wheel the contact between the wheels either create too much friction for the wheel to turn or the contact is too weak (or no contact at all) for several positions of the pinewood wheel. To resolve the issue of slipping, the next iteration involved wrapping adhesive grit around the pine wood wheel to increase friction of contact with the motor wheel. However the bigger issue was still not resolved – imperfections in the pinewood wheel still accounted for occasional slipping due to lack of (or no) contact, and it created too much torque for the motor to turn the wheel for certain positions of the pinewood wheel. The idea at the time of this iteration was to simply use the area of the pinewood wheel that provided consistent actuation performance. However, the idea did not assure 100% consistence in actuation, which is unreliable if the actuation does not function at crucial situations. So for the next iteration, an attempt was made to resolve the issue posed by the imperfections in the pinewood wheel.

The idea with the next iteration was to provide some way to maintain 100% consistent contact between the motor wheel and the pine wood wheel. This was accomplished by replacing the rigid mount of the DC motor by a dynamic one. By fastening (not too tightly) a wooden member to the base by using a wood screw at one end, and mounting the DC motor on the other end, a system is created with a mount (on which the DC motor is) that is allowed to freely rotate about a pivot (the wood screw). 100% consistent contact is then insured by the means of tension from a rubber band wound around two wood screws - one on the mount, the other in another location of the base (the idea can be best visualized in a diagram – see 'figures' section). While this did resolve the issue of inconsistent contact and slipping altogether, another issue was realized with the arm. It turned out an arm fashioned out of pinewood was simply too heavy for consistent actuation (as well as for locomotion of the robot). So for the final iteration, the arm was completely replaced by a new one fabricated from Styrofoam (which is significantly lighter), with structural support to the flimsy Styrofoam provided by a thin wooden strip (for visual representation see 'figures' section.

Future recommendations in improving the design of the arm would be to provide a way for softer stops for the arm (as of the final iteration the 90 degree rotations were stopped by corner braces – needless to say the stop is not gentle).

9.5 – Supporting Calculations

Prior to starting fabrication of the robot (and writing up the proposal) several calculations were done to check feasibility of the design our team was going ahead with. These calculations are as follows:

First to verify the stability of the design, some centre of mass calculations have been performed:

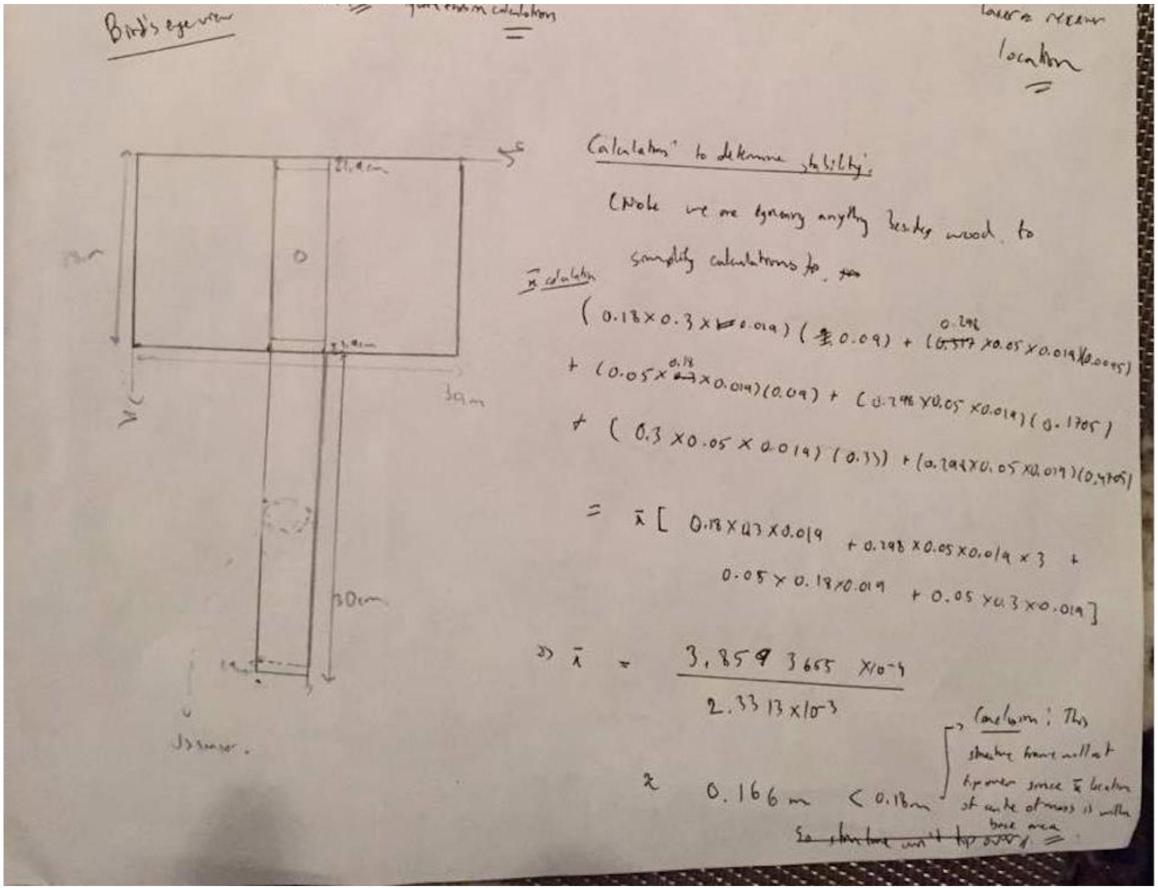


Figure 5: Centre of mass calculations

The x and y-axes were selected to be along two edges of the base. Following this the x co-ordinate of the centre of mass was calculated – if this exceeded 0.18m, then the centre of mass lies beneath the arm which makes the structure unstable and leaves a possibility that the structure may tip over. If the x co-ordinate of the centre of mass is below 0.18m, then the centre of mass still lies within the base, and the structure is stable and would not tip over. The x-co-ordinate of the centre of mass was calculated using –

$$\bar{x} = \frac{\sum M_y}{\sum m}$$

where $\bar{x}$ denotes x-co-ordinate of the centre of mass, M_y denotes the moment exerted by a piece of geometry about the y-axis and m denotes the mass of that piece of geometry. Note that since the structure is made from the same type of wood, it is homogeneous, and the density can be cancelled out from the above equation, simplifying it into a centroid calculation.

Fortunately, the x co-ordinate was calculated to be 0.166m, which is less than 0.18m, making the design chosen stable.

Some torque and motor speed calculations where performed next, to investigate the availability of DC motors that can provide such specs. The moment of inertia of the rotatable structure, which is required to calculate the torque that may be necessary to rotate the structure, is as follows:

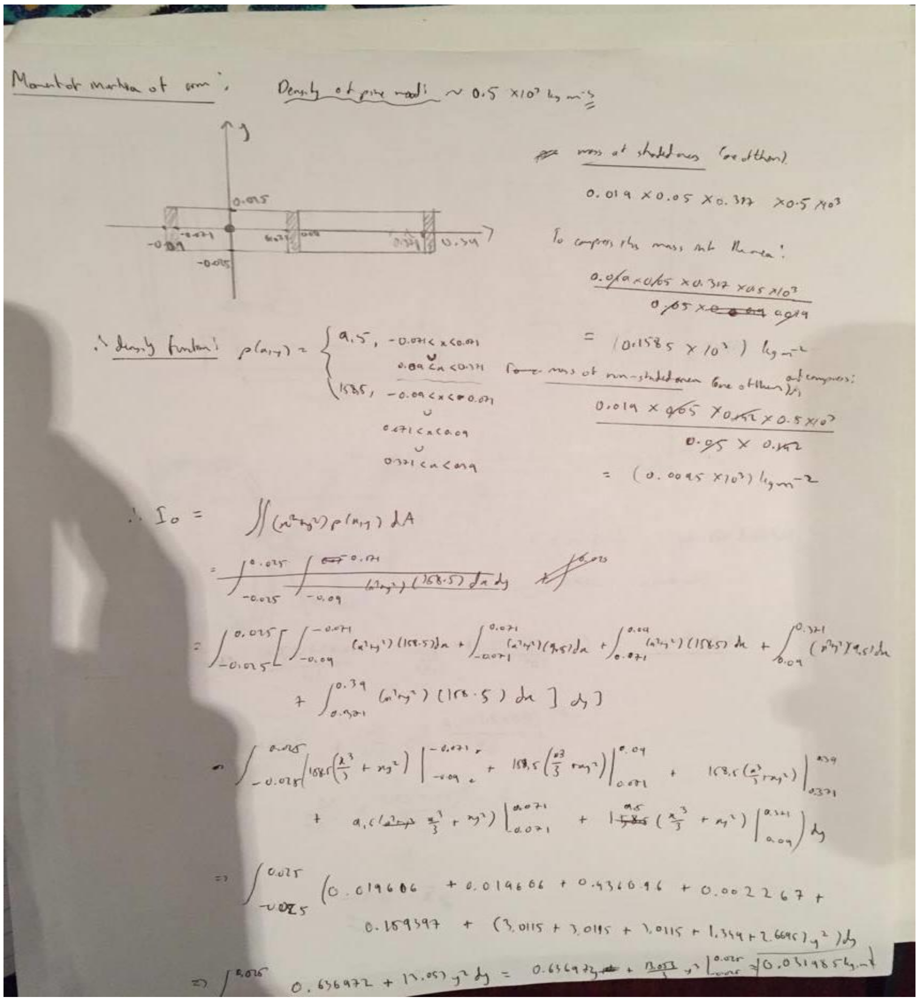


Figure 6 Polar moment of inertia calculations

In the above figure, the centre of rotation was made the origin. The structure has been compressed into a 2D state in the diagram by converting the volume density of wood used to surface density. A surface density function was generated, and then the polar moment of inertia was calculated using the double integral:

$$I_0 = \iint (x^2 + y^2)\rho(x, y)dA$$

where I_0 represents the polar moment of inertia in kg.m^2, and $\rho(x, y)$ represents the surface density function. The moment of inertia of the structure is calculated to be ~0.032 kg.m^2, which can be used in torque requirement calculations.

For calculating the torque required, a simplified model was used and some assumptions were made. The model and assumptions are as follows:

For the retraction mechanism, the arm needs to completely move out of the way the pipe. As an estimate, say the arm needs to move an angular displacement of $\pi/2$ radians. Now, the arm starts at an angular velocity of 0 rad/sec and comes to a complete stop at an angular displacement of $\pi/2$ radians. So both acceleration and deceleration are taking place in between – with the motor accelerating to a certain maximum angular velocity, and then decelerating to a stop. A gradual increase/ decrease in angular acceleration is assumed. To use a simple model, an oscillatory model is assumed, that is the angular displacement, angular velocity and the angular acceleration are assumed to exhibit sinusoidal behaviour.

To start, we know the angular velocity starts and stops at zero. A sine function with a domain from 0 to π can be used to model this. Since it is a sine, we know from Calculus that angular displacement is a negative cosine. Plotting angular displacement against time, we want to place a time that is to be required for this motion. Let's set this time to a good round number, say 10s – that way since this motion has to be performed four times during the operation, it will take only ~40s of the run time. That gives the negative cosine function a period of 20s, and the function it can be modeled by as:

$$\theta(t) = -\frac{\pi}{4}\cos\left(\left(\frac{\pi}{10}\right)t\right), t \,\epsilon [0, 10]$$

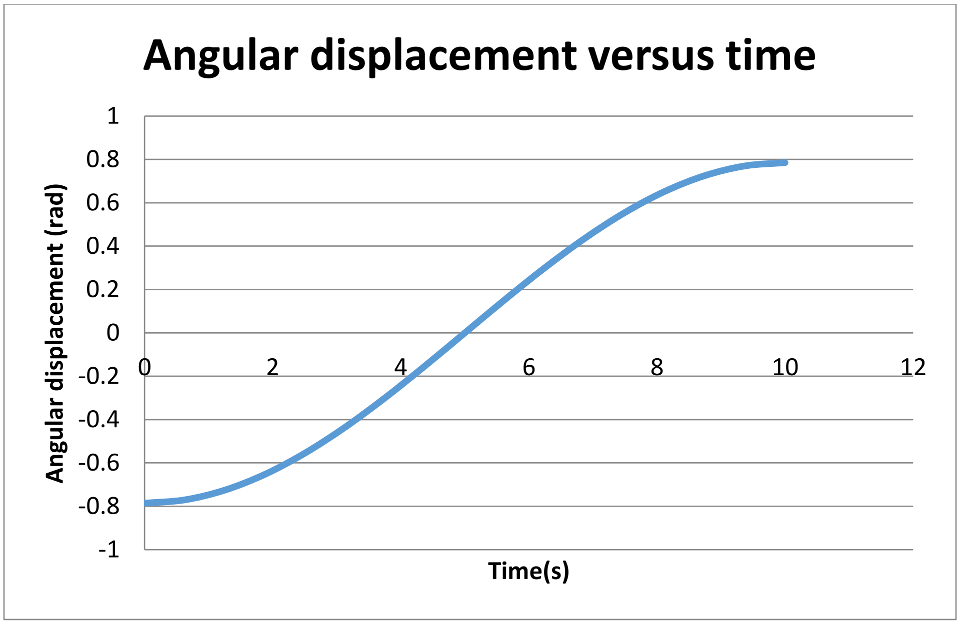


Figure 7: Angular displacement versus time

Differentiating the above function with respect to time, we get angular velocity:

$$\omega(t) = \frac{\pi^2}{40}\sin\left(\left(\frac{\pi}{10}\right)t\right), t \in [0,10]$$

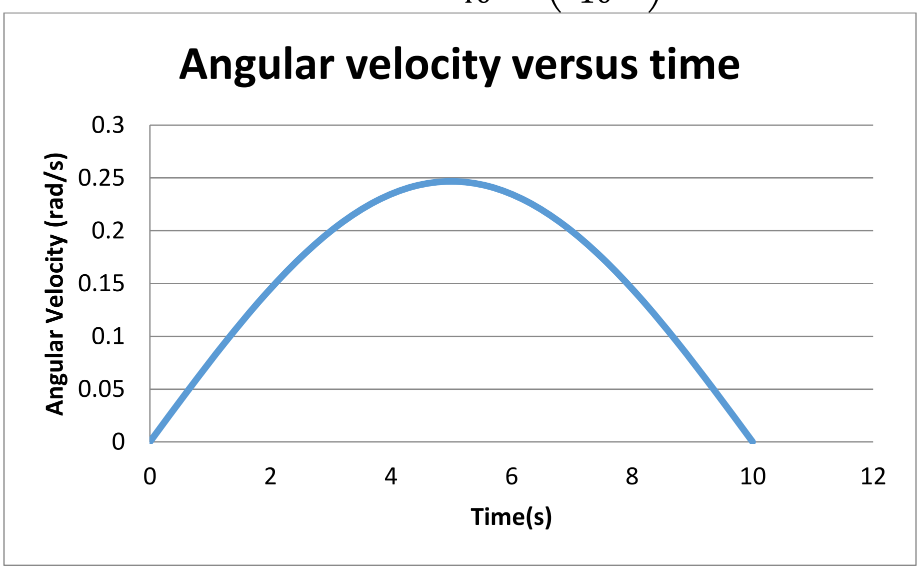


Figure 8: Angular Velocity versus time

Differentiating this will finally give angular acceleration:

$$\alpha(t) = \frac{\pi^3}{400}\cos\left(\left(\frac{\pi}{10}\right)t\right), t \in [0,10]$$

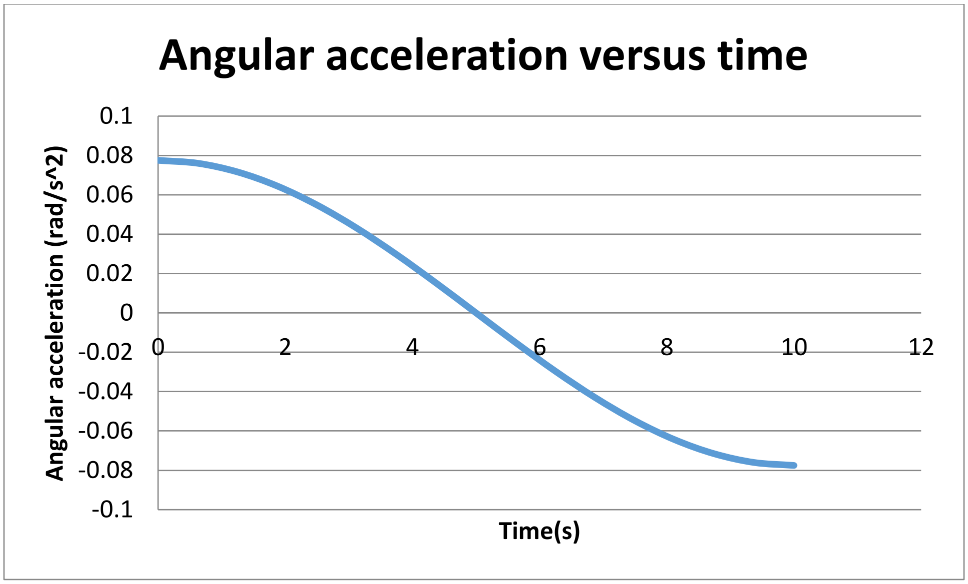


Figure 9: Angular Acceleration versus time

According to this simple model, the motor should be capable of providing an α_{max} of about 0.08 rad/s^2. Using Newton's Second Law of motion for rotation:

$$\tau = I_0 \alpha = 0.032 * 0.08 = 0.00256\ Nm$$

Which is quite low, and a motor can be found that can satisfy this [See electromechanical lecture slide on torque versus r.p.m of DC motors] [3].

N.B. – The above model used is a simple model, and may not give the exact α_{max}. Furthermore, resistive forces where not accounted for when using Newton's Second Law. However, the actual torque should still be in same order of magnitude, and hence the motion is physically possible.

The verification of the availability of motors for the differential drive ought to be performed as well. In order to calculate the torque required of these motors an FBD (i.e. Free Body Diagram) is required:

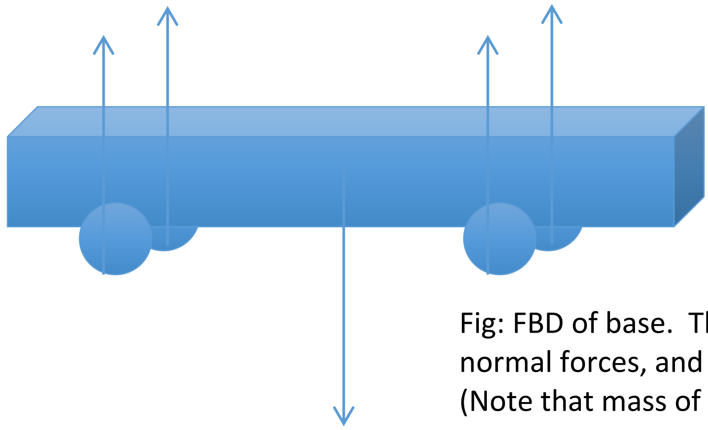


Fig: FBD of base. The upward acting forces are normal forces, and the downward force is weight. (Note that mass of structure is considered here)

The weight of the object can be computed by multiplying the density of the wood (wood used was pine wood, which has density ~500 kg/m^3) with the total volume of structure calculated when doing the centre of mass calculations. One fourth of this is the normal force acting on each wheel which is calculated to be ~2.86 N. The force of friction then during locomotion can be calculated by multiplying this normal force by the co-efficient of force (for floor tiles the standard is ~0.5-0.6 **[3]**) – this gives a friction force of about ~1.57 N at each wheel. The Free body diagram (FBD) seen at each wheel (note that the weight of the wheel is negligible compared to the structure, so we are neglecting this):

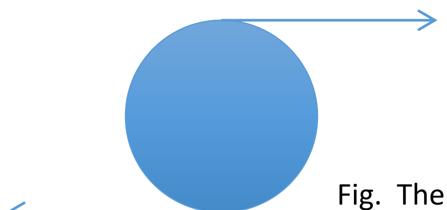


Fig. The normal force is omitted in this FBD. The force acting left is the friction force. By Newton's third law, there is an equal but opposite force acting to the right at the top of the wheel. These forces constitute a couple – the torque being provided by a motor.

The couple can be calculated as:

$$\tau = 2F_f r = 2 * 1.57 * 0.02 = \sim 0.063\ Nm$$

One motor was found in Creatron to provide over ~0.314Nm torque **[3]**, so it can satisfactorily provide this amount. This motor gives a rotational speed of about 40 rpm which translate to going the 400cm distance back and forth in about ~87s. If the time taken for the arm rotation is lowered (the angular acceleration and torque would not increase such that the use of the motor chosen will no longer be valid), then more time will be left for investigating the barrel (even without that we still have a good ~50s for the inspection).

N.B. These calculations were done before the proposal, so naturally it is very backdated related to the present situation (for instance, the centre of mass and stability calculations only involved the wooden frame of the robot; furthermore the motor being used currently for the arm and locomotion is a different one []). It is merely provided as evidence as some feasibility check was performed before going into robot building phase.

Compared to then: Since more components have been added to the base, and a Styrofoam arm has replaced the pinewood arm, stability should not be an issue, seeing as it was not before all of these changes were made. Furthermore since the previous calculations regarding torque required assumed the arm was pinewood, the torque provided by the same motor should be more than enough for present design since Styrofoam is significantly lighter than pinewood. So the only potential area of concern (from a calculations perspective) is the concern that the robot may have gotten too heavy with all the added components for the torque supplied by the DC motors used for locomotion. To verify this is not the case (aside from using experimentation):

$Density\ of\ pinewood, \rho_{pw} = \sim 425\ kg/m^3$ [Toolbox]

$Density\ of\ hardwood, \rho_{hw} = \sim 340\ kg/m^3$ [Data sheet*]

$Density\ of\ plywood, \rho_{plw} = \sim 600 kg/m^3$

[http://www.plywood.cc/2009/03/27/plywood-density/]

$Density\ of\ Styrofoam, \rho_s = \sim 36.5\ kg/m^3$[Wikipedia – for now]

By looking at the member geometries in the ‘figures’ section

Volume of pinewood: $V = (0.30)(0.18)(0.019) + (2)(0.30)(0.05)(0.019) + (0.102)(0.102)(0.019) = \sim 0.001794\ m^3$

Volume of hardwood: $V = \pi(\frac{0.0254^2}{4})(0.28) = \sim 0.000142\ m^3$

Volume of plywood: $V = (0.48)(0.03)(0.002) + \pi\left(\frac{0.1437^2}{4}\right)(0.019) = 0.000337\ m^3$

Volume of Styrofoam: $V = 2(0.316)(0.043)(0.019) = 0.000516\ m^3$

Mass from just frame, $m = \sum \rho V = 1.03kg$

Based on FBD used for locomotion calculations previously, and knowing (from specs) that the chosen motor provides ~0.0981Nm,

The motor wheels are capable of overcoming friction: $F_f = \frac{\tau}{2r} = 3.09N$, which gives a normal force, $F_n = \frac{F_f}{\mu} = \sim 5.62N$, or a total weight of $W = \sim 22.5N$.

So the current motors are capable of moving a robot of ~2.3kg, and since the frame accounts for only less than half of this weight, locomotion is feasible.

9.6 – Tables and Figures

Table 10: Figure 1 labels

Label	Description
1	PIC board fastened to pine columns by using bolts, with nuts functioning as washers. The same method of fastening is used for the other circuits.
2	US Sensor.
3	Thin wood strip as structural support.
4	'Pincer-like' arm.
5	1-inch diameter cavity made in which dowel fits. The arm and dowel are then fastened using a wood screw. The same form is fastening is done with the pinewood wheel and the dowel.
6	Laser. A hole is made in the jaw through which laser fits.
7	Laser receiver circuit.
8	IR transmitter and receiver circuit.
9	1-inch diameter hardwood dowel.
10	Pinewood columns.
11	PIC dev board.
12	Voltage regulator circuit.
13	12V battery holder.
14	Nuts and bolts are used to fasten the pinewood wheel to the top plate of the Lazy Susan. Wood screws are used to fasten the bottom plate of the Lazy Susan to square pinewood mount.

15	Composite Wood wheel. Note that adhesive grit is wound around it.
16	1-1/2 in. corner braces used as stops for the arm.
17	Square pinewood mount.
18	2 in. corner braces to fasten used to attach pinewood columns to base.
19	Woodscrew used to loosely fasten mount to base. Mount freely spins about woodscrew – it is dynamic.
20	Dynamic wooden mount for DC motor.
21	DC motor and wheel set provide actuation to rotate the pinewood wheel.
22	A woodscrew is fastened to the mount while another just a little way up the base. A rubber band is wound the screws – the tension from it ensures the pine wood wheel and the motor wheel are in 100% contact.
23	H-bridge circuit (non-Integrated Circuit (IC)).
24	Nuts and bolts used to fasten Front (rigid*) Castor Wheels to bottom of base.
25	Pinewood base.
26	(Rigid) Castor Wheels.
27	DC motor and wheel set – two of them used as rear wheels to complete the Differential Drive system.
28	Encoder system. Fastened to a mount fashioned out of pinewood in the same way the PIC dev board was. A cavity is present through which wires go to the PIC dev board.
29	IC H-bridge circuit for the DC motors used for locomotion.

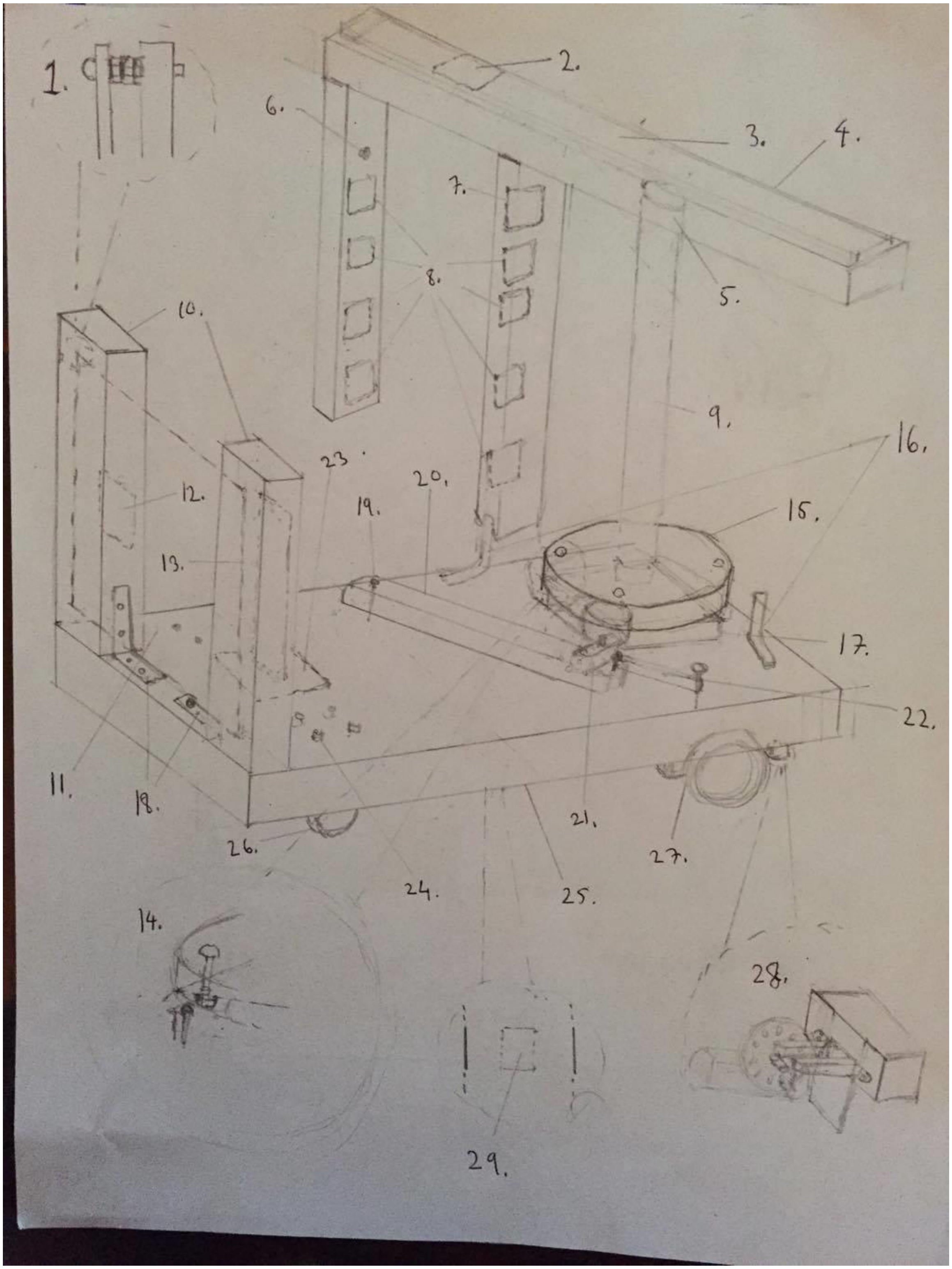


Figure 10: Robot schematic

Table 11: Important specifications regarding the final design

Member	Dimensions
Arm	48cmX4.3cmX1.9cm
Pincers	31.6cmX4.3cmX1.9cm
Dowel	2.54cmX28cm
Columns	30cmX5cmX1.9cm
Motor mount	18cmX1.3cmX1.7cm
Plywood Wheel	14.4cmX1.9cm
Lazy Susan Mount	10.2cmX10.2cmX1.9cm
Base	30cmX18cmX1.9cm
(Locomotion) DC motor plywood mount	4cmX1.5cmX0.7cm
Encoder circuit mount	5cmX1.1cmX1.9cm
Arm structural support	48cmX3.0cmX0.2cm

Table 12: Important components used in final design

Item	Specs
Lazy Susan	Dimensions: 4in.X4in. Load cap: 300lbs.
Hardwood Dowel	Dimensions: 1in.X48in. Weight: 0.4609lbs.
DC gear motor wheel set	Motor rating: 5V. Torque: 1kgcm. Speed: 83rpm.
1-1/2 in. Corner Brace	Dimensions: 1-1/2 in. for each 'leg'. Weight: 0.09lbs.
2 in. Corner Brace	Dimensions: 2 in. for each 'leg'. Weight: 0.17lbs
Rd Soc Stove Bolts (16 pcs)	Thread callout: 1/8X1.
Hexmach Sc Nut Pltd (23P)	Thread callout: 4-40.
Rd Soc Mach Screw (26Pc)	Thread callout: 4-40X1.
Flat Soc Brass Wd Screw (7Pc)	Thread callout: 8X1-1/2.
Flat Soc Brass Wd Screw (9Pc)	Thread callout: 10X1.
Non-slip Adhesive Strip	8' long. 1 in. width.
(Pinewood) Shelving	10in. X 4in. X 1in.
Swivel/Non-Swivel Casters	1-1/4 in. (both). Weight: 0.14lbs

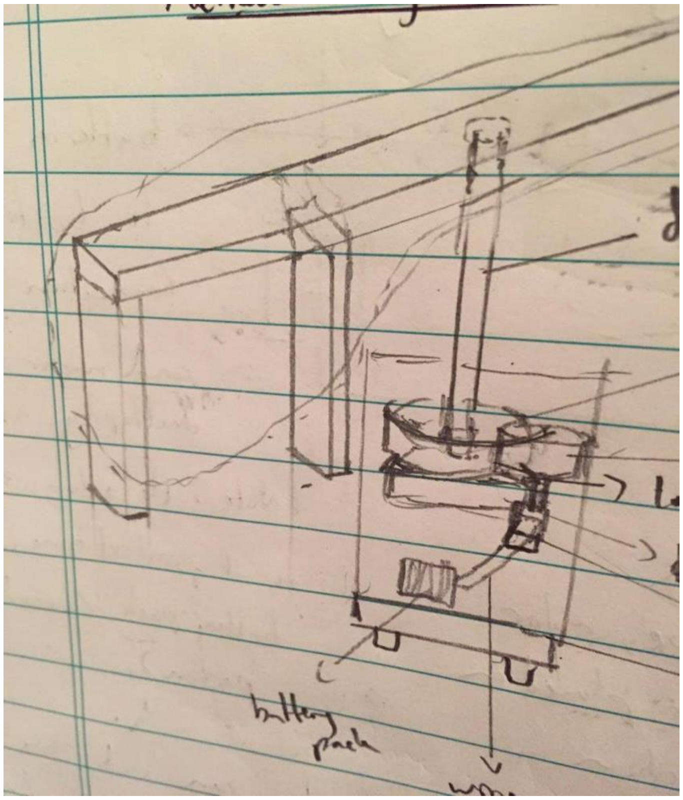


Figure 11:The rigid mount design for the DC motor used in earlier iterations of the arm.

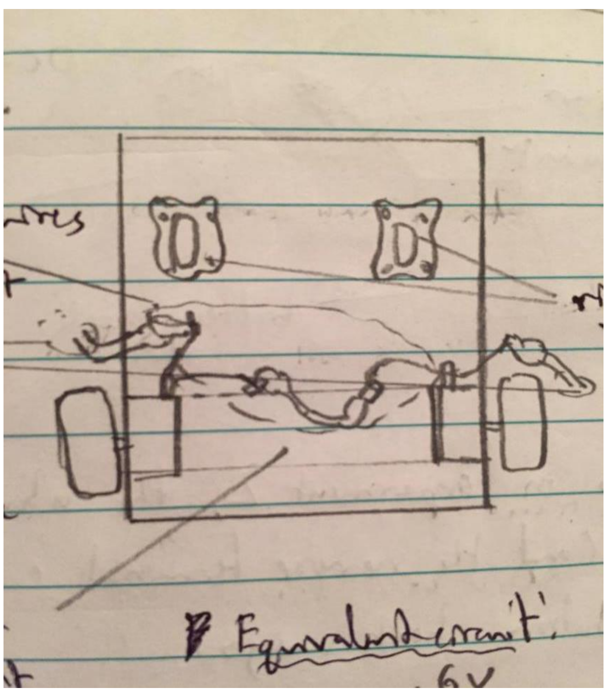


Figure 12: The bottom of the base design for the robot minus the IC H-Bridge circuit.

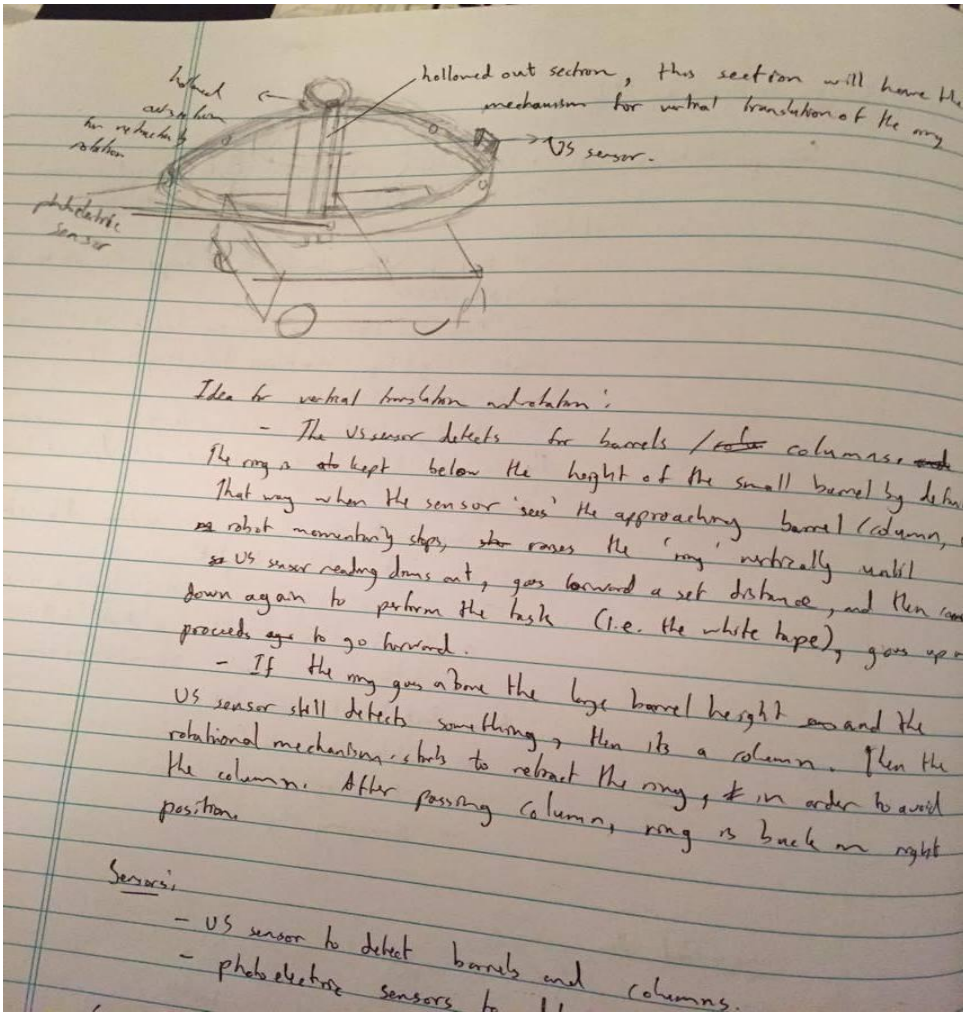


Figure 13: Alternative Design A (see top left of image; ignore the text). The idea is to have a ring with two sensors in the inner lining of the ring, such that they the face the white tape potential positions. The ring can be raised and lowered to investigate the barrel status. In the top position, ring can slide sideways – retraction necessary for avoiding pipe.

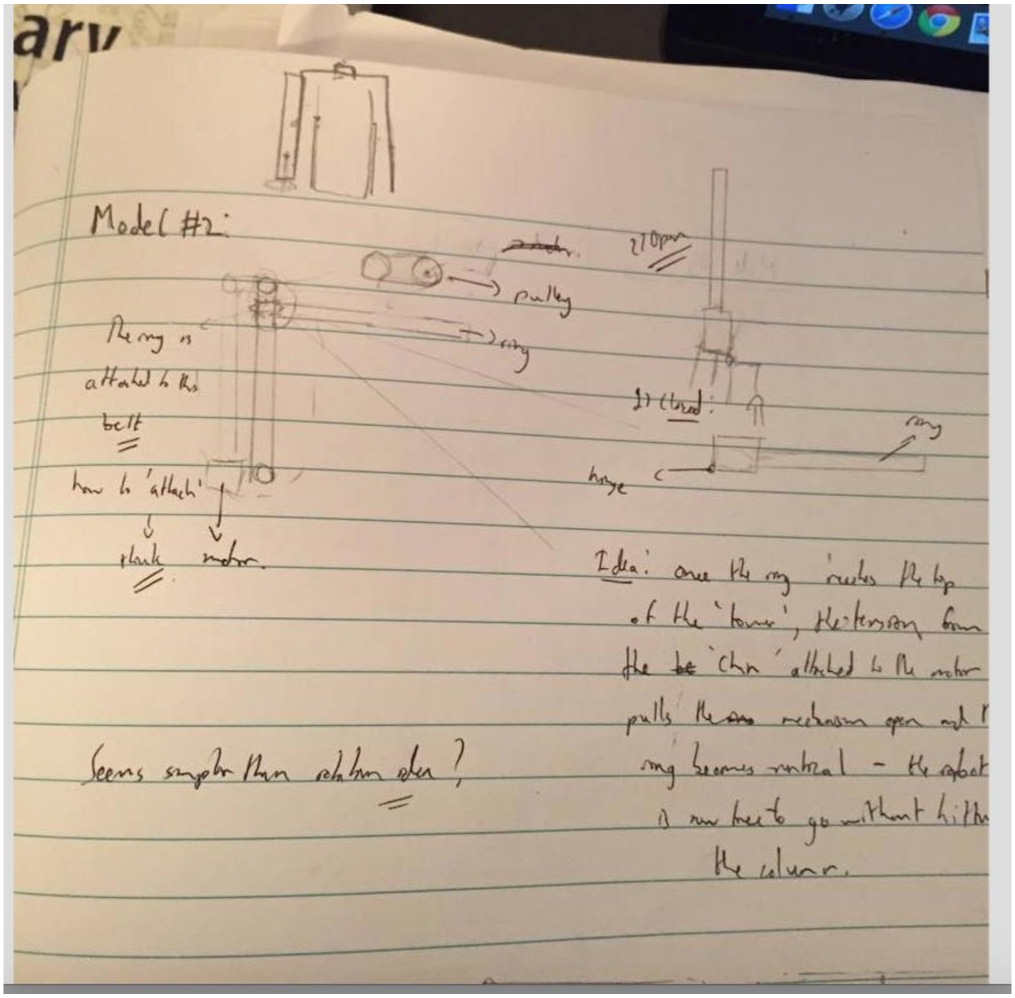


Figure 14: Alternative Design B (ignore the text in image). The ring idea applies here as well. Difference however is that the ring motion up and down as well as retraction is accomplishes using a string-pulley system.

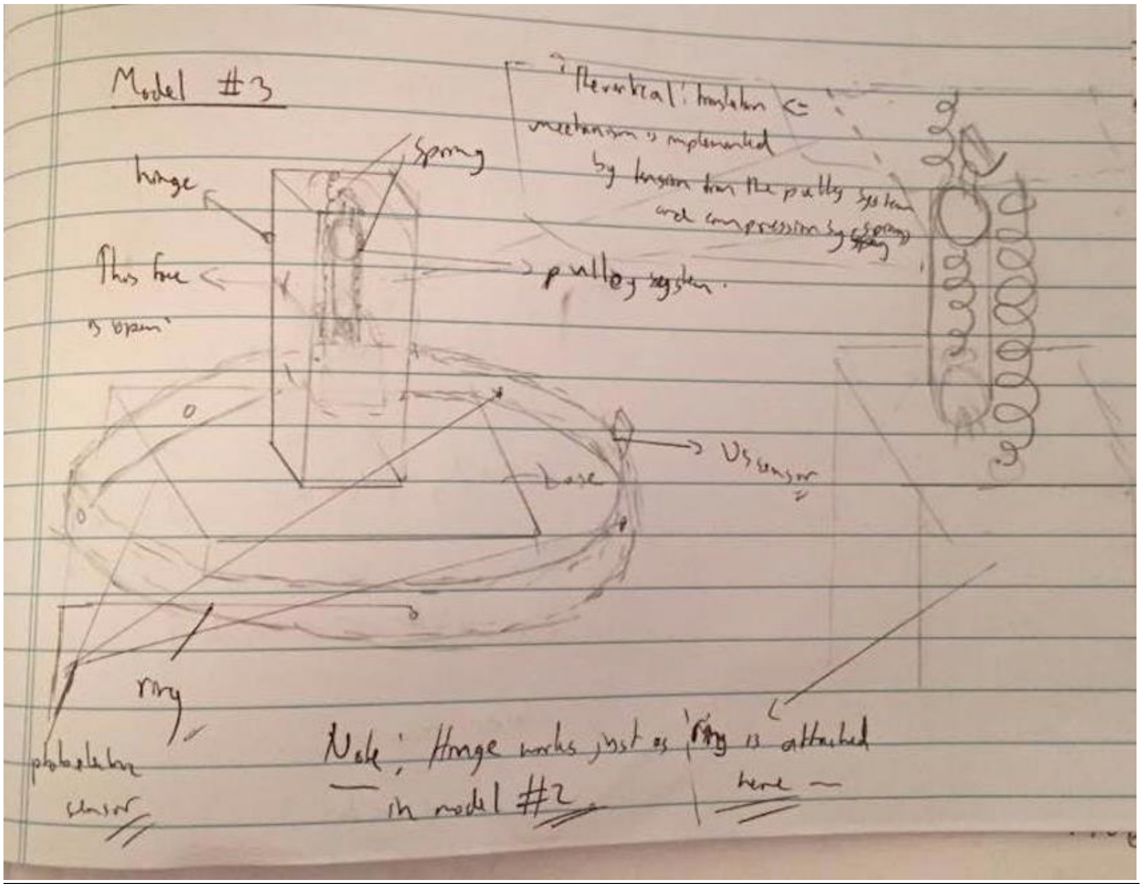


Figure 15 Alternative design C (ignore text in image). This particular design also uses the same ring concept, but uses the tension from string and expansion of compressed springs to elevate and lower the ring respectively.

10. Microcontroller Subsystem

10.1 – Accessing the Problem

The goal of the microcontroller subsystem is to implement a series of algorithms – employing combinatorial and sequential logic – that accepts input from a user and traverses through a linear array of barrels detecting the water level thereof. In particular, the machine is to be activated by a key press on the keypad, and upon completion of the operation, must display a termination message on the LCD displaying inspection information including but not limited to[1], operation time, total number of barrels of each type, the type of each barrel detected, its location with reference to Start Line, and the level of liquid in it.

The problem – the microcontroller task was partitioned to several subsections:

1. **User input detection** that receives a signal from the user to begin operation.
2. **Barrel detection** that polls the sensors to evaluate subsequent decisions.
3. **Barrel type algorithm** that evaluates barrel specifications based on the sensor inputs.
4. **Recording algorithm** that stores information relating to each individual barrel.
5. **Data Analysis** needed to convert the data into readable form.

The subroutines used to accomplish said subsections will be described later in the chapter.

10.2 – Choice of Microcontroller

Of all the numerous microcontroller chips recommended for the project, the PIC16F877 was chosen. Figure X provides the specifications of the PIC16F877 chip along with the PIC18F4620 chip, which will be used as a means of comparison. It can be seen that although the Random Access Memory (RAM) and Electrically Erasable Programmable Read-Only Memory (EEPROM) of the PIC18F4620 greatly exceeds that of the other, the number of I/O ports are about the same. For our purposes, it was determined that memory

[1] The real time clock (RTC) functionality was included, additionally.

offered by the PIC16F877 was sufficient. Furthermore, a smaller instruction set is more convenient to use, in terms of simplicity, usability and debugging. This allows the programming more time to make reliable code and improve readability and efficiency. Coupled with the great pool of resources (documentations) available to the PIC16F877, there is no doubt that it was the ideal microchip. Should the need arise, moving from PIC16F877 to PIC18F4620, or even to parallel processing with the two processors should not be too difficult, as opposed to the other way around. The difference in price between the two chips, $1, is quite substantial should the machine be massed produced – this is an important criteria to take into consideration in industry.

Table 13 Comparing microcontrollers

Microcontroller	PIC16F877	PIC18F4620
Data RAM (Bytes)	368	3968
Data EEPROM (Bytes)	256	1024
Speed MHz	20	40
I/O Ports	33	36
ADC 10-Bit Ports	8	13
Instructions	35	49
Cost	$6	$7

10.3 – Comparator Functions

Unfortunately, unlike most programming languages, the PIC16F877 instruction set is very limited (35 as seen in the above table). This means that, common comparator functions such as "IF" statements, "CASE" statements, or "WHILE" statements are not available. However, these statements can be effectively and equivalently be created using the following functions:

1. **BTFSS f, b** – this function evaluates the **b**-th bit of the register **f**. If it is 1, then the next instruction in program memory is skipped, otherwise, the next instruction in program memory is executed. The usefulness of this function is how certain functions, such as **XORWF** and **SUBWF**, affects certain bits of the **STATUS** register. In particular, said functions affects the **Z** bit and the **C** bit of the **STATUS** register, respectively. Coupling these functions with **BTFSS** brings rise to the common comparator functions.

2. **BTFSC f, b** – this function is similar to **BTFSS**, however, if the **b-**th bit of the register **f** is 0, then the next instruction in program memory is skipped, and if it is 1, then the next instruction in program memory is executed.
3. **GOTO**
    a. If the parameter following the operand is “$” followed by an immediate value or literal X, the program will jump to the next (or previous) X instructions.
    b. If the parameter following the operand is a label, the program will jump to the memory address represented by the label.

10.4 – Pseudocode of Operation

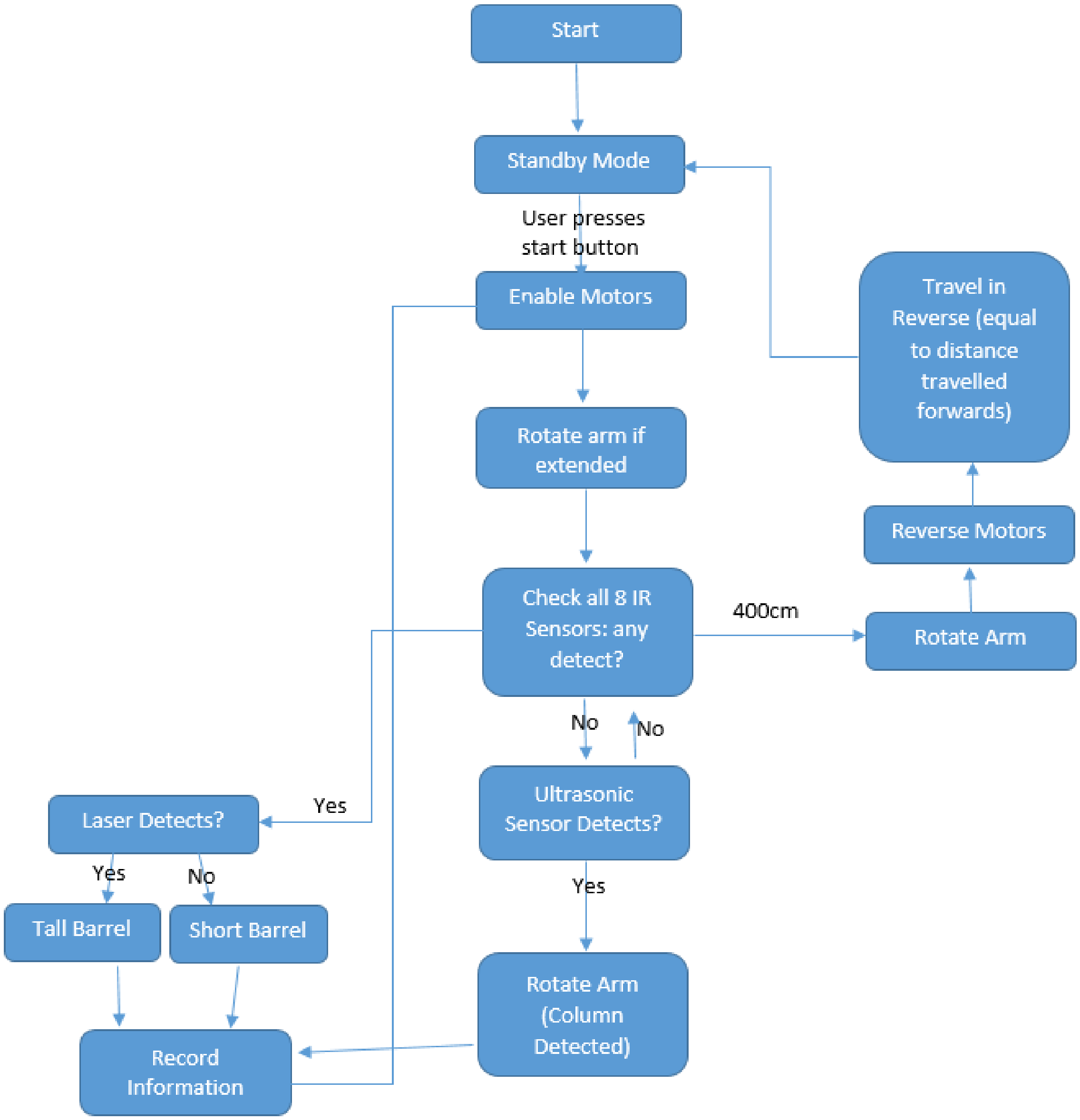


Figure 16: Pseudocode of operation

10.5 – Lookup Table

10.5.1 – Motivation for the Lookup Table

```
Welcome_Msg1
            addwf   PCL,F
            dt              "Welcome!", 0
Welcome_Msg2
            addwf   PCL,F
            dt              "Press * to Start", 0
Message1
            addwf   PCL,F
            dt              "T", 0
Message2
            addwf   PCL,F
            dt              " B:", 0
Message3
            addwf   PCL,F
            dt              "Press * to Reset", 0
Message4
            addwf   PCL,F
            dt              "Press # for Info", 0
Message5
            addwf   PCL,F
            dt              " TP:", 0
Message6
            addwf   PCL,F
            dt              "L:", 0
Message7
            addwf   PCL,F
            dt              "D:", 0
Message8
            addwf   PCL,F
            dt              " OT", 0
```

Figure 17: Lookup table

Due to the way the WR_DATA subroutine is coded, printing to the LCD can require an excessive number of lines. The WR_DATA subroutine takes in a letter stored in the W (working) register, thereby printing each word, letter by letter. Thus, printing a letter requires two lines of code: **1. movlw** “x” **2. call** WR_DATA. The loop-up table, however, provides an efficient solution to this, as shown in Fig. X. Sentences can be stored which can be printed onto the LCD using just one call function.

10.5.2 – Problems with the Lookup Table

The lookup table functions correctly only if its values are stored within the first page (256 bytes) of the program memory. There exists solutions that mitigate this problem, however, when encountered initially, the most efficiency solution was to simply move the entire table to the front of the code, after the register declarations.

10.6 – Problems with variables

Variables are stored in registers in Data Memory, which are stored into four banks, each with 127 registers, respectively. There are Special Function Registers and General Purpose Registers; the former being used for specific purposes such as the **STATUS** register, and the latter being available to the programmer to store information. The registers used to store the variables in the program is from 20h HEX to 7Fh HEX. It is important that the variables do not overwrite a pre-existing Special Function Register, which would cause errors.
The directive **CBLOCK** facilitates the storing of variables as it defines Block Constants. **CBLOCK 0x##** declares the memory address ## on which the user can begin storing variables.

10.7 – Start Operation

```
;*****************************************
; Main code
;*****************************************


Main
            btfss   PORTB, 1
            goto    $-1
            Display Welcome_Msg1
            btfsc   PORTB, 1    ;check if cleared
            goto    $-1
            btfss   PORTB, 1
            goto    $-1
            call    Switch_Lines
            Display Welcome_Msg2

test
            btfss   PORTB, 1    ;check for input from KEYPAD
            goto    $-1         ;if NOT, keep polling

            swapf   PORTB, W    ;when input is detected, swamp nibbles
                                ;PORTB <7-4> moved to <3-0> in w
            andlw   0x0F
            xorlw   b'00001100' ;checks if 12th key is pressed *
            btfss   STATUS, Z   ;if pressed, then Z=1
            goto    test        ;if NOT, then keep checking until * is pressed

            btfsc   PORTB, 1    ;keep iterating until key is released
            goto $-1
            goto START
```

Figure 18: Start operation code

The machine begins operation by displaying welcome messages, and requests the user to press * to begin operation. It does this by polling the PORTB input pins to determine if the user has pressed a button. If the machine detects that the * button was pressed, then it will again poll to see if the * button has been released. This avoids unprecedented activity should the * be held down for a prolonged period.

10.8 – End Operation

```
END_DISPLAY
                call    Clear_Display
                Display Message3
                call    Switch_Lines
                Display Message4

CHECK_PRESS1
                btfss   PORTB, 1     ;check for input from KEYPAD
                goto    $-1          ;if NOT, keep polling
                swapf   PORTB, W     ;When input is detected, read it in to W
                andlw   0x0F
                goto    OPTION1

OPTION1         ;checks if * was pressed
                movwf   option_temp
                xorlw   b'00001100'          ; Check to see if 12th key
                btfss   STATUS,Z             ; If status Z goes to 0, it is the 13th key, skip
                goto    OPTION2          ; If not check if it's B
                call    Clear_Display
                goto    Main               ; If it is, restart

OPTION2         ;checks if # was pressed
                movf    option_temp, W
                xorlw   b'00001110'
                btfss   STATUS,Z
                goto    CHECK_PRESS1 ;resume polling
                call    Clear_Display
                btfsc   PORTB, 1     ;keep iterating until key is released
                goto $-1
                goto    POLL1
```

Figure 19: End Operation code

```
POLL1
                btfss   PORTB, 1    ;check for input from KEYPAD
                goto    Polltime1   ;if no input, poll INFO
                swapf   PORTB, W    ;when input is detected, read it in to W
                andlw   0x0F
                btfsc   PORTB, 1    ;keep iterating until key is released
                goto $-1
                goto    CHECKPRESS1 ;check which key was pressed
Polltime1
                movlw   "T"         ;displays T for Real Time
                call    WR_DATA
                call    Realtime    ;displays Real Time
                Display Message2    ;displays B:
                movlw   "1"         ;displays barrel #
                call    WR_DATA
                Display Message5
                movfw   barrel1 ;T/S/E/HF/F
                call    Check_Type
                call    Switch_Lines
                Display Message6
                movfw   barrel1
                call    Check_Height
                Display Message7
                movfw   barrel1+3 ;ten digit first, how is this stored? Leave as 0's for now
                call    WR_DATA
                movfw   barrel1+2
                call    WR_DATA
                movfw   barrel1+1
                call    WR_DATA
                Display Message8
                call    HalfS
                call    Clear_Display
                goto    POLL1


CHECKPRESS1
BACKWARD1       ;checks if 1 was pressed
                movwf   option_temp
                xorlw   b'00000000'         ;checks to see if "1" was pressed
                btfss   STATUS,Z            ;if status Z goes to 0, it is not "1"
                goto    FORWARD1        ;if not, check to see if "2" was pressed
                call    Clear_Display
                goto    POLL7
FORWARD1        ;checks if 2 was pressed
                movf    option_temp, W
                xorlw   b'00000001'
                btfss   STATUS,Z
                goto    POLL1 ;resume polling
                call    Clear_Display
                goto    POLL2
```

Figure 20: Display code

Once operation is finished, the machine is ready to communicate with the user the barrel results. Again, the machine polls the PORTB pins: if * is pressed, then it restarts operation and resets all barrel information stored, and if # is pressed, then the machine displays each barrel information, one at a time, starting from the first barrel. After displaying all information pertaining to a specific barrel, it will again poll the PORTB pins: if “1” is pressed, it will retrieve information pertaining to the last barrel, and if “2” is pressed, it will retrieve information pertaining to the next barrel. This code snippet is instantiated eight times for each of the respective barrels.

10.9 – Main Operation

Referring back to the 1.4, the pseudocode for the operation, the program first checks if 400 cm has been traversed; if it has, then it will turn back. Otherwise, it will poll the eight IR sensors rapidly, if any of the IR sensors detects anything, it will then check the laser sensor which is placed above the height of the small barrel. If the laser sensor detects anything, it is then a tall barrel, and if it doesn’t, then it is a short barrel. If the last IR sensor doesn’t detect anything, it will then check the ultrasonic sensor for a column. If it does detect, it will rotate the arm out of the way. The eight IR sensors were strategically placed, and their respective locations are shown earlier. There are four on each side of the barrel, four on each arm. This is because the white tape could be on either side of the barrel.

10.10 – Subroutines

10.10.1 – IR Sensor Code

```
CHECK_IR1
                ;initialize IR_DETECT
                movlw   b'0'
                movwf   IR_DETECT

                movlw   b'11000001' ;to select IR1
                movwf   ad_store
                movfw   ad_store
                call    IR_MAINLOOP
                return ;GO BACK TO OPERATION CODE
```

Figure 21: IR sensor code

The following code checks if the first IR sensor detects anything. IR_DETECT is the register that stores a 0 if it doesn't detect, and a 1 if it does detect. Line 1076 is passed to the ADCON register which selects which analog port (out of the eight) to check. It is also stored in the ad_store register so that this literal value is not lost as it goes to check the other IR sensors. This code snippet is instantiated eight times for each of the IR sensors.

```
1152  IR_MAINLOOP
1153            movfw   ad_store ;storing the ADCON value
1154            call    AD_CONV
1155            movwf   voltage_IR
1156            call    CHECK_IR
1157            btfss   STATUS,C
1158            goto    DISPLAYCHECKHIGH
1159            goto    DISPLAYLOW
1160            return
1161            ;C is set when voltage >= 4.1
1162  CHECK_IR
1163            movlw   b'10111101'
1164            subwf   voltage_IR,W
1165            return
```

Figure 22: IR main loop

The IR_MAINLOOP subroutine calls AD_CONV, which converts the voltage across the sensor into a digital value, via an analog-to-digital converter. The IR sensor converts infrared radiation into voltage, and the threshold value for which the sensor detects "white" in contrast with "black" was determined by connecting a voltmeter across the sensor. This value is then converted to a binary number or Analog Digital Converter (ADC) value through the following equation:

$$\text{ADC value} = \frac{\text{Voltage}}{5} \cdot 256$$

The "5" represents the amount of voltage supplied the sensor, which normalizes the voltage value detected across the sensor. It is then converted to a binary number from 0 to 256, where an ADC value of 0 and 256 would represent voltages of 0V and 5V, respectively. Each time the IR sensor is polled, it will return an ADC value which is compared with the threshold value, which in this case was 4.1V. As mentioned earlier, the **SUBWF** will set the **C** bit of the **STATUS** register under certain conditions, and by setting **C=0** when the voltage is under the threshold, it can be determined when "white" tape is detected. The IR sensor uses a negative-true logic, in which case the voltage drops when it detects “white”, so that is why it is the voltage *under* the threshold that is important.

It was observed that random fluctuations in the voltage would occur, and sometimes the **C** bit would trigger due to such random fluctuations. To ensure that there is in fact “white” detected when the **C** bit is triggered, and not due to random noise in the surroundings, the following subroutine, INCREMENT_IR, is used. Only when there are four consecutive triggers does the subroutine acknowledge that a barrel is detected. Note that these triggers occur on the scale of microseconds, as instructions in program memory occur at that scale, so it is not likely that the barrel would “miss” the white tape as it travels along.

```
INCREMENT_IR
                movfw   counter_IR
                xorlw   d'4'
                btfsc   STATUS,Z
                goto    DISPLAYHIGH
                incf    counter_IR
                movfw   counter_IR
                xorlw   d'4'
                btfsc   STATUS,Z
                goto    DISPLAYHIGH
                goto    IR_MAINLOOP
                ;return
```

Figure 23: Increment code

10.10.2 – Ultrasonic Sensor Subroutine

```
;****************************************
; ULTRASONIC SENSOR CODE
;****************************************


ULTRASONIC
                movlw   d'16'  ;initialize timer module
                movwf   T1CON
                movlw   d'0'
                movwf   TMR1L
                movlw   d'0'
                movwf   TMR1H

                bsf     US_TRIG ;10us TRIGGER HIGH
                call    DelayL
                bcf     US_TRIG ;TRIGGER LOW

                btfss   US_ECHO ;waiting to detect echo (HIGH)
                goto    $-1
                bsf     T1CON, 0 ;turn timer on / TMR1ON=1
                btfsc   US_ECHO ;waiting for echo to go LOW
                goto    $-1
                bcf     T1CON, 0 ;turn timer off

                movfw   TMR1L
                movwf   Time_Low
                movfw   TMR1H
                movwf   Time_High

                return
```

Figure 24: US sensor code

The ultrasonic sensor operates by sending a 10 microsecond trigger (US_TRIG goes to high and back to low) and waits for an echo (US_ECHO goes to high). When it detects an echo, it will start a timer, T1CON, until the echo fades (US_ECHO goes to low). The time length of the echo is stored in two registers, with the higher order bits in Time_High, and the lower order bits in Time_low. Instead of directly computing the distance of an object away from the sensor, which requires complicated calculations involving the speed of sound in air, the sensor was tested by outputting the timer values onto the LCD. It was estimated that the barrel would be around ~5 cm from the sensor itself, and that threshold value, as a timer value was recorded. The higher order bit of the timer value was 3, and is used in the CHECK_US subroutine (Fig. X) to compare the current timer value with the threshold timer value. If the current timer value is less than the threshold value, that means there is an object closer than 5 cm.

```
CHECK_US
                call    ULTRASONIC
                ;C is set when Time_High > 3
                movlw   b'11' ;3
                subwf   Time_High,W
                btfss   STATUS, C
```

Figure 25: Checking Ultrasonic

10.10.3 – Encoder

In order to implement the encoder, a subroutine Distance_Count was used in order to increment the distance each time the encoder sensor encounters a division. The Distance_Count function has a ones, tens, hundreds, and thousands. The number of divisions on the wheel was counted – 20 – and the circumference of the wheel was found to be 26.98 mm after three measurements. That is 4.238 mm per division, which was rounded to 4.25 mm per division as the difference is insignificant over the total distance of 400 cm. Since it was not possible to store decimal numbers on the PIC, by multiplying 4.25 by 4, a whole number, 17, was obtained. Hence, every four clock cycles of the ultrasonic sensor, the Distance_Count would increment by 17 mm. That is, every four detections of the sensor, 17 calls to Distance_Count would be made. When the Distance_Count reaches 4000 mm, or equivalently 400 cm, the machine would stop and turn around. Appropriate delays were required as the program would double count the distances if the wheel stayed stationary at a division “hole”. The following is the code for the encoder:

```
OPERATION_ENCODER
                movlw   b'0' ;reset the counter
                movwf   dis_counter
                ;call   Clear_Display
                btfss   ES ;check if ES gets a HIGH
                goto    OPERATION_ENCODER
                ;call   DISTANCECALL17

                ;check if 4 divisions are counted
                incf    dis_counter4
                movfw   dis_counter4
                xorlw   d'4'
                btfss   STATUS, Z ;Z=1 when dis_counter4=4
                goto    OPERATION_ENCODER
                call    DISTANCECALL17
                goto    OPERATION_ENCODER


    DISTANCECALL17
                    movlw       b'0'
                    movwf       dis_counter4
                    movfw       dis_counter
                    xorlw       d'17' ;Z=1 if dis_counter=17
                    btfsc       STATUS,Z
                    return ;if Z=1, return
                    call        Distance_Count
                    incf        dis_counter
                    goto        DISTANCECALL17
```

Figure 26: Encoder

10.10.4 – Pulse Width Modulation and Motors

It was found that the DC motors operated too quickly, so Pulse Width Modulation (PWM) was employed to lower the voltage seen through the motor. It operates by changing the amount of time the voltage is HIGH or LOW, effectively changing the average value of the voltage seen by the motor. The percentage or proportion of “on” time to “off” time is called the Duty Cycle. The Duty Cycle was set to 80% due to the weight of the machine. If the Duty Cycle was any lower, the machine would not move. A Duty Cycle of 0% would correspond to an open circuit, no voltage being sent through, and a Duty Cycle of 100% would correspond to a closed circuit, full voltage being sent through.

```
;****************************************
; MOTOR SUBROUTINE (PWM)
;****************************************

MOTOR_ON_RC1
                movlw   b'11111111'
                movwf   CCPR2L
                bsf     PORTC, 1
                return
MOTOR_ON_RC2

                movlw   b'11111111'
                movwf   CCPR1L

                bsf     PORTC,2
                return
MOTOR_BOTH_OFF
                movlw   b'00000000'
                movwf   CCPR2L
                movwf   CCPR1L
                return


        ;100% 11111111 255
        ;80%    11000111 199
        ;60%    10010101 149
        ;0%             00000000
```

Figure 27: Motor Code

As seen in the above figure, turning the motor on and off is not as simple as simply sending a HIGH or LOW. In order to turn on the motor, the duty cycle, converted to a binary number, is sent to the **CCPR1L/CCPR2L**, each corresponding to one of the two PWM pins.

$$\text{Duty Cycle (CCPR1L/CCPR2L)} = \frac{\text{Binary Number}}{255} \cdot 100\%$$

In order to turn off the motors, a Duty Cycle of 0% is sent through both motors. CCPR2L was used for the left motor and CCPR1L was used for the right motor.

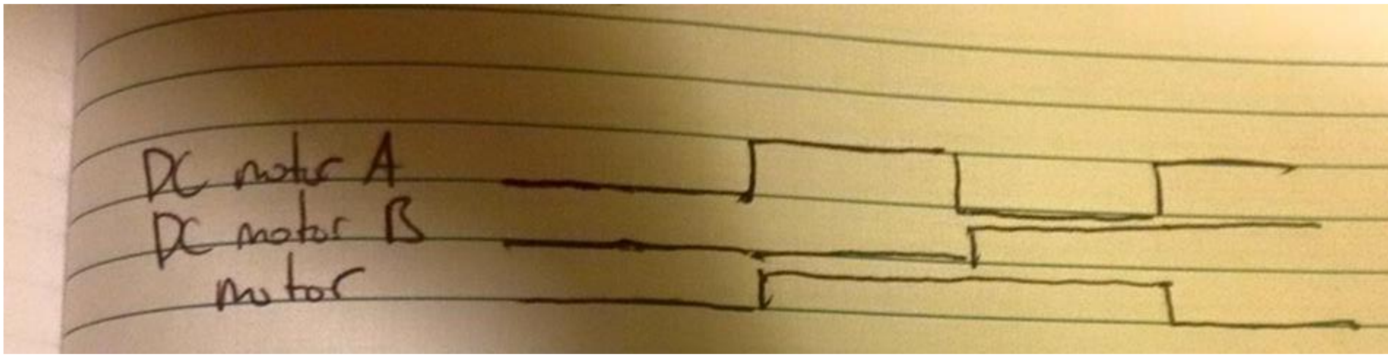


Figure 28: Timing diagram for motors

The H-Bridge contains to outputs to the PIC, and the motor is turned on only when one of the outputs is HIGH and the other is LOW, or vice-versa. Fig. X illustrates the timing diagram for each motor, where DC motor A and DC motor B corresponds to the two output pins to the motor, and "motor" represents the actual state of the motor.

10.10.5 – Barrel Identification

The following is the pseudocode for identifying the specifications of the barrel:

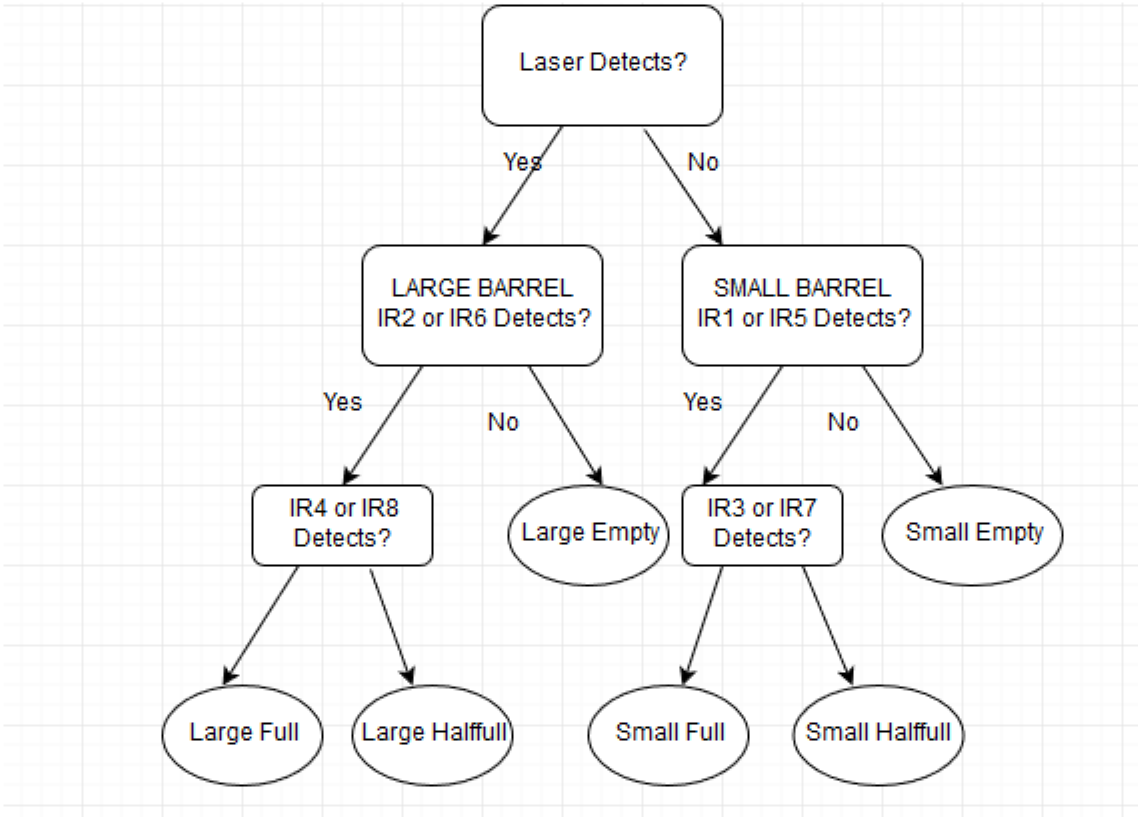


Figure 29: Barrel detection flowchart

10.10.6 – Record Subroutines

Once the barrel specifications are known, it is stored in the register barrel_data, as seen in Fig. X. One register is enough to completely specific the barrel information, using the convention (S=1/T=0)/E/HF/F corresponding to the bits 3/2/1/0. For example, if the barrel is Small and Empty, then the binary number stored in barrel_data is 00001100.

```
;*****************************************
; MOVE INFO TO BARREL_DATA SUBROUTINE
;*****************************************

RECORD_SE
                movlw   b'00001100' ;(S=1/T=0)/E/HF/F
                movwf   barrel_data ;store it so it can be recorded
                goto    RECORD
RECORD_SHF
                movlw   b'00001010'
                movwf   barrel_data
                goto    RECORD
RECORD_SF
                movlw   b'00001001'
                movwf   barrel_data
                goto    RECORD

RECORD_TE
                movlw   b'00000100'
                movwf   barrel_data
                goto    RECORD
RECORD_THF
                movlw   b'00000010'
                movwf   barrel_data
                goto    RECORD
RECORD_TF
                movlw   b'00000001'
                movwf   barrel_data
                goto    RECORD
```

Figure 30: Recording subroutine

Once the barrel information is stored in barrel_data, it is passed into the record subroutine into the corresponding barrel number.

1.9.7 – Real Time

The real time functionality is made possible by using a battery and short-circuiting the two I2C pins. The following is the code for the real time clock:

```
Realtime
                rtc_read        0x01            ;Read Address 0x01 from DS1307---min
                movfw   0x77
                call    WR_DATA
                movfw   0x78
                call    WR_DATA
                movlw                   ":"
                call    WR_DATA

                ;Get seconds
                rtc_read        0x00            ;Read Address 0x00 from DS1307---seconds
                movfw   0x77
                call    WR_DATA
                movfw   0x78
                call    WR_DATA
                return

                call    OneS                    ;Delay for exactly one seconds and read DS1307 again
                goto    show_RTC
```

Figure 31: RTC code

10.11 – Issues with the PIC

One of the main issues with the PIC was that sometimes the board would not boot up properly. There would be white spaces on the top row of the LCD display. Almost all the time, de-attaching the BUS on the PIC would resolve the problem, suggesting that one of the wires on the BUS was short circuiting causing the PIC to not boot up. Other times, cleaning the program memory using the picUSB software would resolve the problem.

10.12 – Pin Assignments

Table 14: Pin Assignments

PORT	PIN	I/O	A/D	Function
A	RA0-3, 5	Input	A	Infrared Sensors
B	RB1, RB4-7	Output	D	4x4 Keypad
C	RC0-1	Output	D	DC motor A1/2
	RC2	Output	D	DC motor B1
	RC5	Output	D	Ultrasonic Trigger
	RC6	Input	D	Ultrasonic Echo
D	RD0	Input	D	Encoder Sensor
	RD2-3	Output	D	HD44780 LCD
	RD4, 6	Output	D	DC motor C2/1

	RD5	Output	D	DC motor B2
	DC7	Input	D	Laser Sensor
E	RE0-3	Input	A	Infrared Sensors

11. Circuits Subsystem

11.1 – Problem Assessment

When one accepts the role of the circuits' subsystem, they might be a bit confused when it comes to the requirements. There isn't a given guide and they are not sure how to go about taking the first step. The true potential of the circuits is revealed when one realises that circuit subsystem is meant to be the communicational control between the electromechanical subsystem and the microcontroller subsystem. For the Barrel Inspector Project there are three main requirements that the circuits' member is to design and implement: The Drivetrain of the Robot, The Water Level Detection, The Column Detection and rotation of the arm, and The Barrel Type Detection.

The project requires the entire operation to be completed under three minutes else ten points will be subtracted for each during the operation. In addition, the power delivery to the drivetrain should be smooth so that there are no current spikes or lag that may prove to be an issue during the operation. The drivetrain control circuits should be able to provide variations in the current delivery so that the speed of the robot is able to be controlled by the microcontroller. Finally the drivetrain system should have an area for input from the microcontroller which provides electrical signals.

The project requires the robot to approximately measure the height of the barrels. The tapes that are placed on the barrels are white and are placed at approximately 2/3 and 1/3 of the barrels' heights. If the tape detected is above 2/3 the height then the barrel is assumed to be full, if it is between 2/3 and 1/3 then the barrel is assumed to be half full and finally if it is below 1/3 then the barrel is assumed to be empty. Because there is a large contrast between the white tape and the black barrel, the detection system has to be able to detect the difference

The project requires the robot to detect the presence of a column of indefinite height that is placed somewhere between the barrels and avoid it. The robot must be able to find the column and know its distance from the column and when to avoid it. The most common ideology that one may have is to rotate the arm out of the columns way as adjusting the position of the entire robot is more difficult and out of the scope of second year students.

Finally the robot must be able to differentiate between large barrels and small barrels. The small barrel has maximum height equivalent to 2/3 of the large barrel. The robot must be able to notice the difference in height and record this in its memory.

11.2 – Solutions

11.2.1 – Drivetrain Control

When it comes to rotation of the arm and the moving of the robot a bipolar H-bridge is needed to be able to reverse the polarity of the DC motors. The project requires that at least one of the actuators be manually made. The project does not however restrict the team to only use open circuit actuators, therefore integrated circuits can also be used. For the rotation of the arm, a bipolar H-bridge was made according to the schematic below. For the actual drivetrain system the IC SN754410NE Quad half H-Bridge was used. To limit the rotation speed of the arm a 5 V input was sent into the Bi-Polar H-Bridge instead of a 12 V. However as the actual moving of the robot requires a larger leeway when it comes to speed control a 12 V input was used for the integrated circuit implemented. The open circuit design was taken from the AER201 book under the electromechanical section. The benefits of this design is its simplicity to code as well as the efficiency of its components. The Transistors TIP147 and TIP142 are capable of running up to 10 A of current through them. The arm is to stop once rotated one direction and once again in the opposite direction. This causes the motor to stall and a DC motor can draw up to 8 A of current at stall so a high current rating is needed to ensure the circuit does not fail. Finally the N4001 diodes are there to prevent reverse flow of current which will damage the circuit. The choice of arm rotation over arm retraction was made to avoid using linear railways and servos alongside gearboxes as this increases the cost of the arm itself.

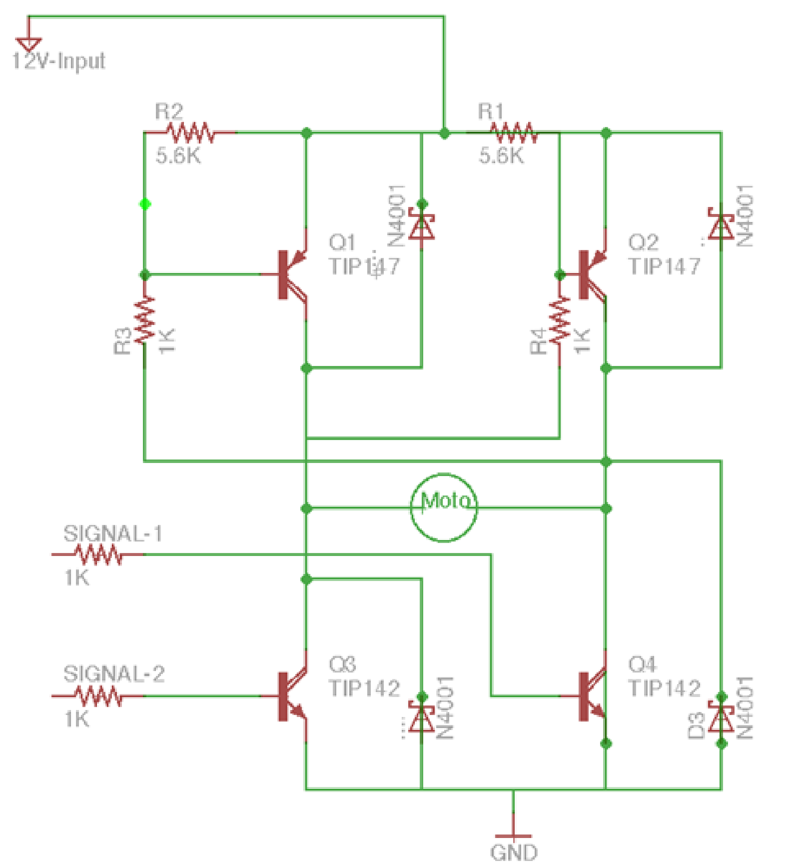


Figure 32 H-bridge

11.2.2 – Water Level detection

It was decided that eight IR sensors to be used for the detection of the water levels. There will be four IR sensors on both sides of the arm and they are placed at 2/3 of the large barrels' height, 1/3 of the large barrels' height, 2/3 of the small barrels' height, and 1/3 of the small barrels' height. The following is the schematic used for the IR circuits. The LTE-4208A and the PT1504-6B can detect 940nm on the electromagnetic spectrum and are ideal in luminescent lighting. The output to the PIC microcontroller comes after the PT1504-6B before circuit finishes. This circuit has the ability to differentiate between white and black surfaces. The white surfaces reflects light much more than the black surfaces and so when there is a white surface present the IR receiver (PT1504-6B) doesn't allow much current to pass and so the value sent is very low (approximately < 0.4 V). When a black surface is present, the IR receiver allows most of the current to pass and the value sent is high (approximately 3.9 – 4.1 V). This being the case, we use all eight analog ports in the PIC microcontroller development board and set a threshold of 3.9V. If the input voltage is below the threshold then the white tape is present and water level detection begins. The PIC checks all the IR sensors simultaneously and can record the height of the water based on the number of responses received. It should also be noted that IR emitter and receiver have a 40 degree emit and receiving span respectively. This allows full detection of tape under or over the approximate heights.

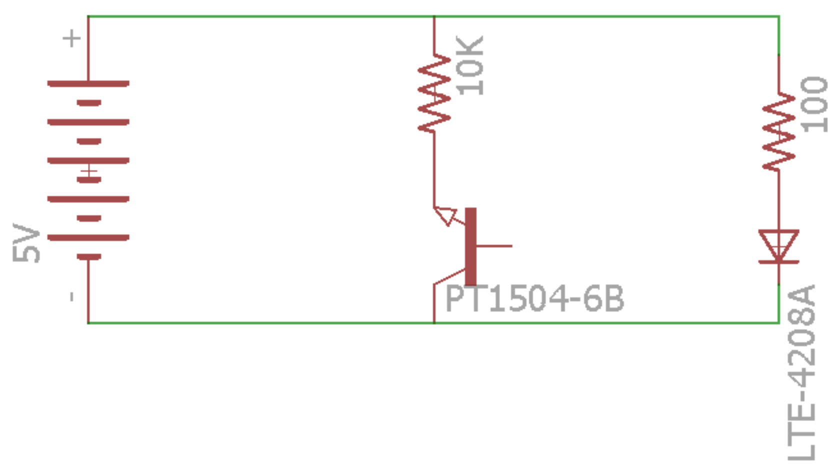


Figure 33: IR sensor

11.2.3 – Column Detection

To detect the column an ultrasonic sensor was bought. This is the HC-SR04 Ultrasonic Sensor which costs approximately $10. This particular sensor was bought because of its ability to detect anywhere from 2cm to 400cm. Since the entire operation can be only 400cm at a maximum, this sensor is the most ideal as it allows us to recognize the column from the start and keep track of the robots distance from the column at all times and when the distance between the robot and the column is that which is desired, the arm will turn. The following is a picture of the ultrasonic sensor. The echo and trigger ports are connected to the PIC, the VCC is given a 5V and the GRN is grounded to the common ground.

Figure 34: US sensor

11.2.4 – Barrel Type Detection

For the detection of the type of barrel it was thought that a break beam circuit would be most optimal solution in order to be able to detect the difference in barrel type. Since no break beam circuit exists that spans 30 cm, a custom laser break beam circuit was made. The following is the schematic of the circuit. Since all the analog ports have been taken, this circuit must be digital and have outputs of either high or low strictly.

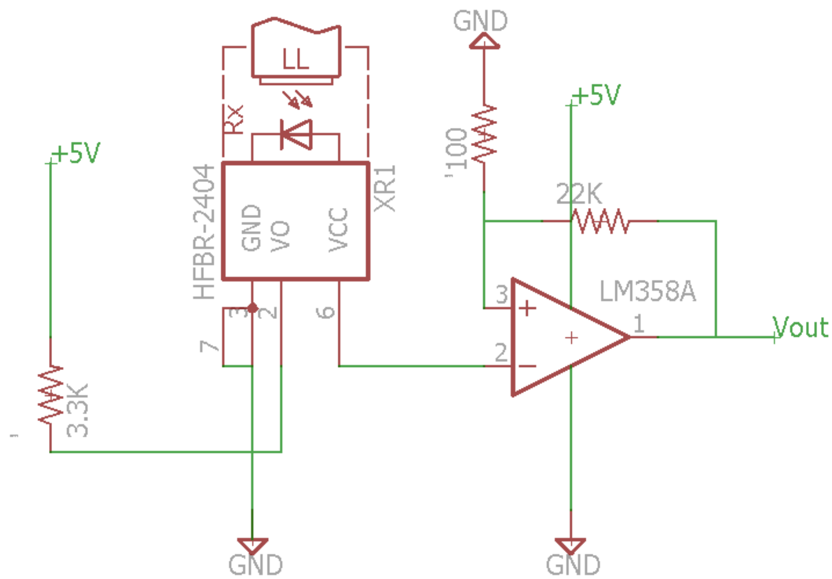


Figure 35: Laser receiver circuit

When the laser makes contact with the receiver the signal sent to the PIC microcontroller is a low which is a 0 V. When the laser beam is broken the signal sent is a high which is approximately 3.7 V. This is a large enough difference for the Microcontroller to tell the difference between Low and High signals. In the coding process by checking both the IR sensors and the laser beam, it can be detected whether the barrel being inspected is a large barrel or small. The laser is placed at a position where only the tall barrel can "break". If the IR sensors are giving a signal and the laser is broken then the barrel is a large else it is a small.

11.2.5 – Voltage Regulator

The power supply used in the robot provides 12V but some of the circuits use 5V and so there must be some regulation involved. The IR, Ultrasonic, and Laser circuits however had their own power supplies of 5V. The regulator was only used for the H-bridges that required 5V inputs. The following is the schematic of the voltage regulator created. The capacitors added are not required since the entire project used DC current but they were added as a precaution. The 7805 Regulator converts 12V to 5V by dissipating the difference in voltage as heat

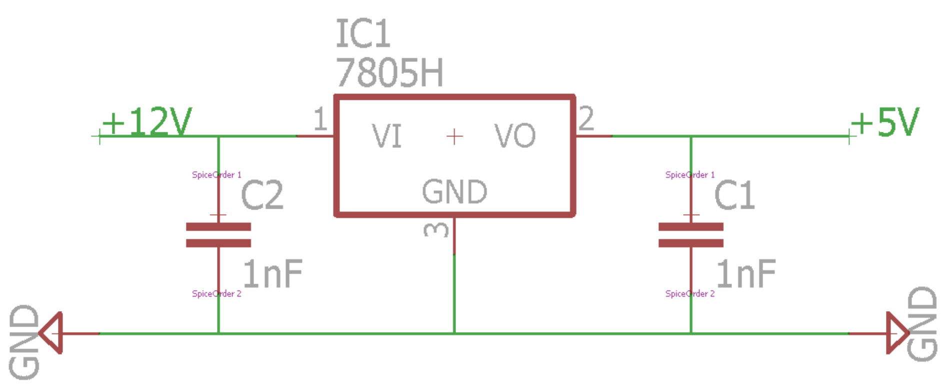


Figure 36: Voltage regulator

11.3 – Testing/Troubleshooting

All of the circuits that were made were tested manually first to ensure they work properly. They were made by first testing on breadboards then once they were confirmed to work to satisfaction, they were soldered onto PCB boards. Once again they were tested to make sure soldering was done correctly and that mistakes in the connections were not made.

11.3.1 – H-Bridge Testing

When testing the open circuit H-bridge actuator it was noticed that the transistors would heat up and the circuit would fail to work after some time without reason. It was found out that the transistors were overheating and as such to fix the problem Heat Sinks were attached to each transistor. The IC H-Bridge was a bit more difficult to get working, the IC has two ports on each side that is attached to PIC and one is allowed to have a high while the other is to have a low. Trying to simulate this scenario was a bit difficult as the IC works with digital signals not analog. It was later found that by grounding one of the ports, it is the equivalent as sending a Low signal. Besides of these little issues the testing of the H-bridges went smoothly. The power was sent to the H-bridge, the motor outputs were attached to the motor terminals.

11.3.2 – IR Testing

The IR testing went smoothly from the start without problem. The circuit that was designed in the schematic was created and tested. The Voltage was read across the receiver and the difference seen was to satisfaction. It was noticed that the IR sensors could read up to 20cm when positioned horizontally. With such results, the design was confirmed and implemented fully onto the robot.

11.3.3 – Laser Circuit

The laser circuit was one that gave many problems when it came to testing. Firstly it was noticed that the circuit would give negative voltage signals when the voltage was measured, this was later solved when it was found out that the ground and power pins were mistakenly connected. Afterwards, the circuit gave the problem that the voltage being measured would not change when the laser beam was being broken or realigned. It was also noted that when touching he back of the PCB board, the connections that would be connected through the person's body would show a single high voltage with no regard to presence of the laser. This problem was resolved when it was noticed that there was a loose connection and to have this resolved resulted in the High and Low difference that was to be expected.

12. Integration

The integration phases of the machine will be discussed herein, from the beginning phases leading up to the Project Demonstration.

12.1 – 03/19/2016

The initial phase of integration involved running the motors with the physical machine. It was found that the wooden arm was too heavy to rotate properly or at all, and adding a counterweight rendered the DC motors useless. As even on full power, the DC motors were not strong enough to move the machine. It was decided upon that a Styrofoam arm is used instead to reduce the weight of the structure. At this point, we discussed the needs of slowing down the motor using PWM. IR sensors were measured precisely and fastened onto the wooden arm by means of screws.

12.2 – 03/20/2016

Problems were experienced with the laser sensor, its ability to detect was a function of its position and orientation. The PIC board was then fastened on to the front of the machine on two pieces of wood protruding from the base. To test the hypothesis of a lighter arm, the entire arm was detached and it was seen that the base rotation still exhibited "jutting" behavior – the movement was not smooth. In fact, it was seen that clockwise rotation was fast and smooth, whereas only the counter-clockwise rotate exhibited slow and "jutting" behavior. We found there was an optimal regime on the base where the rotation was smooth irrespective of direction, and whose sector is 90 degrees of the total circumference. This is sufficient as the robot need not rotate fully 360 degrees, but only a

90 degree sector for arm rotation. During this day, the arm rotation was tested successfully moving back and forth.

12.3 – 03/28/2016

The entire robot was disassembled in order for the construction of the new Styrofoam arm. It was noticed that even with the new arm, there was still a problem with rotation. This time, it was not due to the weight of the arm itself, but the unevenness or inexactness of the circumference of the circle base. An elastic band was found to be too tight, hindering the motion of the arm. The location of the elastic band was placed in closer proximity to reduce the tension of the elastic band.

12.4 – 04/07/2016

Due to complications with the design and wiring, integration was delayed until the day before the Project Demonstration. Minimal testing was conducted, and while each individual component of the robot was found to work (the IRs functioned correctly, the ultrasonic likewise, etc), only the motors, arm rotation and the ultrasonic were functioning correctly together. Further testing revealed that the robot moved in circular motion, rather than straight.

12.5 – System Improvement Suggestions

Many improvements with the machine can be suggested, with the first relating to batteries. It was found that the batteries drained consistently fast for no apparent reason. The use of rechargeable batteries would be more cost-efficient. Even at the end, with the Styrofoam arm, the entire machine was still heavy in the sense that the full power of the DC motor was required in order to have the machine move. Even with a Duty Cycle of 80%, the machine was struggling to move forward. Improvement in the machine wiring would needed as most of the time, connections came lose which could potentially be dangerous. Too much time was wasted in soldering said connections together, and by

Murphy's Law, the machine could not operation during Project Demonstration due to a connection to the power came loose. This resulted in boot-up problems with the PIC, as wire(s) connected to the PIC BUS were short-circuiting due to unstable wire connections. Lastly, micro-switches should have been used for the arm rotation so that the rotation could be made precise, and a physical multiplexer (MUX) would have been beneficial in order to allow the arm to have PWM.

13. Conclusions

The aim of this project was to develop a fully-functional proof-of-concept of an autonomous robot capable of investigating heavy water levels in nuclear power plants. In response to the posed RFP, our team developed a robot using a Differential Drive with the intention to drive (as of the end) completely straight in order to avoid upsetting barrels or colliding with the pipe obstacles arranged in a linear row. A simple design with as few as required components as possible was developed in order to keep the design process free of complexity and ensure final weight and cost of product was as minimal as possible. Finally, the arm is designed in a pincer-like shape, with IR sensors and Laser receiver circuit mounted on the jaws to 'detect the liquid level' and identify the barrel type – the pincer-like shape ensuring both the sides of the barrel facing toward the robot and away from it are covered. Finally a US sensor is placed on top of the arm to detect oncoming pipe obstacle, and necessary actuation provided to retract arm by rotation when facing it (the mechanism to return it back to 'liquid measuring position' and keeping it retracted are obviously in place as well, depending on the situation). Finally, the results of the inspection are shown in the LCD screen on the PIC dev board fastened to the front of the robot.

However, our robot could not be completed within the time set for the project as we failed to show overall functionality by the end. It was mainly owing to a large chunk of time spent debugging our respective subsystem and leaving no time at the end for proper debugging and full integration.

Although, if time was available, our team did draft up some suggestions for improving out system which is elaborated in the System Improvement Suggestions. As to improvements in the project, trying not-so-linear paths in the future would be enterprising. After all, pragmatically speaking it is unlikely the barrels will be arranged exactly in a row, and allowing for adjustment of direction on this basis could be the next area of study. Furthermore this could be extended to investigating barrels placed in multiple rows to make the functionality more ubiquitous. In the event of trying this new direction, the budget would be most constraining (just based on our project $230CDN is hardly enough). Maybe allowance for more budget would enable this direction of study.

14. References and Bibliography

[1] K. D. Peterson, "The Stored Waste Autonomous Mobile Inspector (SWAMI)," 27 01 2016. [Online]. Available: http://www.iaea.org/inis/collection/NCLCollectionStore/_Public/26/033/26033370.pdf.

[2] UCDavis, "Standard Normal Distribution," 27 01 2016. [Online]. Available: https://www.math.ucdavis.edu/~soshniko/135a/materials/standardnormaltable.pdf.

[3] Johnsonite, "Specifications, Product Testing and Terminology PDF," Johnsonite, [Online]. Available: http://www.johnsonite.com/TechnicalData.aspx. [Accessed 28 01 2016].

[4] Creatron, "GM8 143:1 Gear Motor (3V 40RPM)," Creatron, [Online]. Available: https://www.creatroninc.com/product/gm8-143-1-gear-motor-3v-40rpm/. [Accessed 27 01 2016].

[5] N/A. [Online]. Available: [2] UCDavis, "Standard Normal Distribution," 27 01 2016. [Online]. Available: https://www.math.ucdavis.edu/~soshniko/135a/materials/standardnormaltable.pdf.. [Accessed 08 04 2016].

15. Appendixes

CODE

```
	list p=16f877			; list directive to define
processor
	#include <p16f877.inc>		; processor specific variable
definitions


	__CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC &
_WRT_ENABLE_ON & _CPD_OFF & _LVP_OFF
    #include <rtc_macros.inc>

    cblock  0x20
        COUNTH
        COUNTM
        COUNTL
        Table_Counter
        lcd_tmp
        lcd_d1
        lcd_d2
        com
        dat ; was in sample code
        count   ; used to convert optime to decimal for display
        ones    ; ones digit of the converted binary number
        tens    ; tens digit of the converted binary number
        huns    ; hundreds digit of the converted binary number
(hopefully not used)
        binary_num ; move optime to this variable to allow binary -->
decimal for display
        w_temp  ; saves the value in the working register
        status_temp ; saves the current state of the status register
(for ISR)
        barrel1:4
        barrel2:4 ;41
        barrel3:4 ;44
        barrel4:4 ;47
        barrel5:4 ;50
        barrel6:4 ;53
        barrel7:4 ;56
        ;1: Stores Tall/Short Barrel, Stores E/HF/F
        ;2: Location (stores distance < 256 cm)
        ;3: Location (if distance > 256 cm)
        barrelnum   ;current barrel number
        barreltemp
        option_temp
        Delay1
        Delay2
        TIMCNT
        voltage_IR
        counter_IR
        lastop_IR ;checks last operation of IR
        IR_DETECT
        Time_High
```

```
        Time_Low
        ad_store
        dis_counter ;increments to 17 for the encoder
        dis_counter4 ;increments to 4 before incrementing 17 times for
the encoder
        min:2   ;temporary registers for operation time
        sec:2
        initmin:2
        initsec:2
        finalmin:2
        finalsec:2
        armextend ;set to 1 when arm extended
        Dis_Ones
        Dis_Tens
        Dis_Hunds
        Dis_Thous
        ultra_time
        threshold_time
        barrel_data
    endc

    cblock  0x70

    COUNTH1 ;const used in delay
    COUNTM1 ;const used in delay
    COUNTL1 ;const used in delay

    endc


    ;Declare constants for pin assignments (LCD on PORTD)
        #define RS   PORTD,2
        #define E    PORTD,3

        ;ANALOG PINS
        #define IR1  PORTA,0
        #define IR2  PORTA,1
        #define IR3  PORTA,2
        #define IR4  PORTA,3
        #define IR5  PORTA,5
        #define IR6  PORTE,0
        #define IR7  PORTE,1
        #define IR8  PORTE,2

        ;DIGITAL PINS
        #define DCA1    PORTC,1 ;DC motor A1 PWM
        #define DCA2    PORTC,0 ;DC motor A2
        #define DCB1    PORTC,2 ;DC motor B1 PWM
        #define DCB2    PORTD,5 ;DC motor B2 ;A7
        #define DCC1    PORTD,6 ;DC motor C1
        #define DCC2    PORTD,4 ;DC motor C2

        #define US_TRIG PORTC,5 ;Ultrasonic TRIGGER
        #define US_ECHO PORTC,6 ;Ultrasonic ECHO

        #define LS   PORTC,7 ;Laser Sensor Bottom - Digital
        ;#define    LSH PORTD,1 ;Laser Sensor Top - Digital
```

```
        #define ES  PORTD,0 ;encoder sensor
         ORG        0x0000     ;RESET vector must always be at 0x00
         goto       init       ;Just jump to the main code section.

;DCB???

;***************************************
; Delay: ~160us macro
;***************************************
LCD_DELAY macro
    movlw   0xFF
    movwf   lcd_d1
    decfsz  lcd_d1,f
    goto    $-1
    endm


;***************************************
; Display macro
;***************************************
Display macro   Message
        local   loop_
        local   end_
        clrf    Table_Counter
        clrw
loop_   movf    Table_Counter,W
        call    Message
        xorlw   B'00000000' ;check WORK reg to see if 0 is returned
        btfsc   STATUS,Z
            goto    end_
        call    WR_DATA
        incf    Table_Counter,F
        goto    loop_
end_
        endm

bank0   macro
    bcf STATUS, RP0
    bcf STATUS, RP1
    endm
bank1   macro
    bcf STATUS, RP0
    bsf STATUS, RP1
    endm
bank2   macro
    bsf STATUS, RP0
    bcf STATUS, RP1
    endm
bank3   macro
    bsf STATUS, RP0
    bcf STATUS, RP1
    endm
binconv macro
    movwf   binary_num
    call    BIN2BCD
    movf    huns,W
    call    WR_DATA
```

```
    movf    tens,W
    call    WR_DATA
    movf    ones,W
    call    WR_DATA
    endm


;****************************************
; Initialize LCD
;****************************************
init
         clrf      INTCON         ; No interrupts

;   bsf        INTCON, GIE   ; enable global interrupts
;   bsf        INTCON, 5     ; enable timer 0 interrupts
;   bcf        INTCON, 4     ; clear timer0 interrupt flag
;   bcf        INTCON, 2     ; disable internal interrupts (from Port B)
;   bcf        INTCON, 1     ; clear internal interrupt flag.

     ; NEED TO FIX THESE SETTINGS
        bsf        STATUS,RP0     ; select bank 1
    movlw     b'00101111'    ; set RA4 as output
        movlw     b'11111011'    ; Set required keypad inputs
        movwf     TRISB
        clrf      TRISC          ; All port C is output
              ;Set SDA and SCL to high-Z first as required for I2C
    bsf    TRISC,4
    bsf    TRISC,3
    bsf    TRISC,7
    bsf    TRISC,6   ;US_ECHO
        clrf      TRISD
    bsf    TRISD,0 ;the encoder is an input
    bsf    TRISD,1

    movlw     b'00000111'    ;set RE0-3 as input for IR sensors
    movwf     TRISE

        bcf       STATUS,RP0     ; select bank 0
        clrf      PORTA
        clrf      PORTB
        clrf      PORTC
        clrf      PORTD
        clrf      PORTE


    ;Set up I2C for communication
    call    i2c_common_setup
    ;rtc_resetAll

    ;Used to set up time in RTC, load to the PIC when RTC is used for
the first time
    call      set_rtc_time

    call      InitLCD     ;Initialize the LCD (code in lcd.asm;
imported by lcd.inc)
```

```
; Set up Pulse Width Modulation (PWM)
    bsf         STATUS,RP0          ; Bank1
    movlw       b'11111001'         ; Configure PR2 with 10 kHz
    movwf       PR2
    bcf         STATUS,RP0          ; Bank0
    movlw       b'00001111'         ; Configure RC1 and RC2 as PWM
outputs
    movwf       CCP2CON             ; RC1
    movwf       CCP1CON
    movlw       b'00000100'         ; Configure Timer2
    movwf       T2CON               ; Set to prescaler 1:1, postscaler
1:1 , enabled
    movwf       T1CON

    ; Initialize motor variable
    clrf        CCPR2L              ; Set RC1 to 0% duty cycle
    clrf        CCPR1L
    ;bcf        PORTB,0
    ;bcf        PORTC,0
    ;bcf        PORTC,2
    ;bsf        PORTC,5
    ;bcf        PORTC,6



;****************************************
; Main code
;****************************************


Main


        btfss   PORTB, 1
        goto    $-1

        Display Welcome_Msg1

        btfsc   PORTB, 1    ;check if cleared
        goto    $-1

        btfss   PORTB, 1
        goto    $-1

        call    Switch_Lines

        Display Welcome_Msg2

test
        btfss   PORTB, 1    ;check for input from KEYPAD
        goto    $-1     ;if NOT, keep polling

        swapf   PORTB, W    ;when input is detected, swamp nibbles
                    ;PORTB <7-4> moved to <3-0> in w
        andlw   0x0F
        xorlw   b'00001100' ;checks if 12th key is pressed *
        btfss   STATUS, Z   ;if pressed, then Z=1
```

```
        goto    test        ;if NOT, then keep checking until * is
pressed

        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto START


;***************************************
; Look up table
;***************************************



Welcome_Msg1
        addwf   PCL,F
        dt      "Welcome!", 0
Welcome_Msg2
        addwf   PCL,F
        dt      "Press * to Start", 0
Message1
        addwf   PCL,F
        dt      "T", 0
Message2
        addwf   PCL,F
        dt      " B:", 0
Message3
        addwf   PCL,F
        dt      "Press * to Reset", 0
Message4
        addwf   PCL,F
        dt      "Press # for Info", 0
Message5
        addwf   PCL,F
        dt      " TP:", 0
Message6
        addwf   PCL,F
        dt      "L:", 0
Message7
        addwf   PCL,F
        dt      "D:", 0
Message8
        addwf   PCL,F
        dt      " OT", 0

;***************************************
; OPERATION CODE
;***************************************


START
        call Clear_Display


        ;initializing

        ;intialize barrel1/2/3/4/5/6/7
```

```
movlw   b'01011000' ;ASCII X
movwf   barrel1
movwf   barrel2
movwf   barrel3
movwf   barrel4
movwf   barrel5
movwf   barrel6
movwf   barrel7

;intialize barrel1/2/3/4/5/6/7 + 1

movlw   b'00100011' ;#
movwf   barrel1+1
movwf   barrel2+1
movwf   barrel3+1
movwf   barrel4+1
movwf   barrel5+1
movwf   barrel6+1
movwf   barrel7+1
movwf   barrel1+2
movwf   barrel2+2
movwf   barrel3+2
movwf   barrel4+2
movwf   barrel5+2
movwf   barrel6+2
movwf   barrel7+2
movwf   barrel1+3
movwf   barrel2+3
movwf   barrel3+3
movwf   barrel4+3
movwf   barrel5+3
movwf   barrel6+3
movwf   barrel7+3

movlw   d'0'
    movwf   barrelnum
movwf   barreltemp


movlw   b'00110000'
movwf   Dis_Ones
movlw   b'00110000'
movwf   Dis_Tens
movlw   b'00110000'
movwf   Dis_Hunds
movlw   b'00110000'
movwf   Dis_Thous


movlw   d'0'
movwf   IR_DETECT
movwf   threshold_time
bsf STATUS, C ;preset C to 1
movlw   b'0'
movwf   Time_High
movlw   b'0'
movwf   Time_Low
```

```
        movwf   dis_counter
        movwf   dis_counter4



;check if ALl the IRs work
;TEST_IR
;       call    CHECK_IR1
;       movfw   IR_DETECT
;       call    CHECK_DETECT
;
;       call    CHECK_IR2
;       movfw   IR_DETECT
;       call    CHECK_DETECT
;
;       call    CHECK_IR3
;       movfw   IR_DETECT
;       call    CHECK_DETECT
;
;       call    CHECK_IR4
;       movfw   IR_DETECT
;       call    CHECK_DETECT
;
;       call    Clear_Display
;       goto    TEST_IR
;
;
;CHECK_DETECT
;       xorlw   b'1'
;       btfss   STATUS,Z
;       goto    SHOWLOW
;       call    SHOWHIGH
;CHECK_EXIT
;       return
;
;SHOWHIGH
;       movlw   '1'
;       call    WR_DATA
;       return
;
;
;SHOWLOW
;       movlw   '0'
;       call    WR_DATA
;       goto    CHECK_EXIT
;
OPERATION_ENCODER
        movlw   b'0' ;reset the counter
        movwf   dis_counter
        ;call   Clear_Display
        btfss   ES ;check if ES gets a HIGH
        goto    OPERATION_ENCODER
        ;call   DISTANCECALL17

        ;check if 4 divisions are counted
        incf    dis_counter4
        movfw   dis_counter4
```

```
        xorlw   d'4'
        btfss   STATUS, Z ;Z=1 when dis_counter4=4
        goto    OPERATION_ENCODER
        call    DISTANCECALL17
        goto    OPERATION_ENCODER

    DISTANCECALL17
            movlw   b'0'
            movwf   dis_counter4
            movfw   dis_counter
            xorlw   d'17' ;Z=1 if dis_counter=17
            btfsc   STATUS,Z
            return ;if Z=1, return
            call    Distance_Count
            incf    dis_counter
            goto    DISTANCECALL17
;
OPERATION_START


        ;call   Realtime


        ;call   CHECK_DISTANCE
        ;btfsc  STATUS, C   ;if not set, then continue
        ;goto   END_OPERATION   ;if set, then turn back

        ;btfss  ES  ;check if Encoder Sensor detects
        ;goto   START1; if doesn't detect, continue
        ;call   Distance_Count  ;if so, increment the Distance



START1
        ;DISTANCE DISPLAY FOR DEBUGGING ONLY

        ;movlw  " "
        ;call   WR_DATA
        ;movfw  Dis_Hunds
        ;call   WR_DATA
        ;movfw  Dis_Tens
        ;call   WR_DATA
        ;movfw  Dis_Ones
        ;call   WR_DATA
        ;call   Clear_Display

        ;DISTANCE DISPLAY FOR DEBUGGING ONLY

        ;movfw  armextend
        ;xorlw  b'0'
        ;btfss  STATUS,Z
        ;goto   RETRACT_ARM_BACK ;else, retract arm
```

```
        ;at this point, it is clear we have no obstructions, turn on
the motors
        ;turn on the left motor, turn on the right motor
        call    MOTOR_ON_RC1
        call    MOTOR_ON_RC2


        ;*************TEST*
        ;call   RETRACT_ARM_BACK ;just to see the arm rotate back and
forth with delay of 1 second
        ;*************TEST*

        ;we continue to operate until any of the 8 IR sensors detects
something and
        ;at the same time, we are detecting if there is a column w/ the
ultrasonic sesnor
        ;this is effectively a VERY FAST poll that checks between the
IR sensors and the ultrasonic sensors
CHECK_IRSENSORS

        ;checks all 8 IR sensors

        call    CHECK_IR1
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR2
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR3
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR4
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR5
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR6
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR7
        movfw   IR_DETECT
        call    CHECK_DETECT

        call    CHECK_IR8
        movfw   IR_DETECT
        call    CHECK_DETECT

        ;if none sensors detected, check US sensor
        goto    CHECK_US
```

```
CHECK_DETECT
        xorlw   b'1'
        btfss   STATUS,Z ;Z=1 if IR_DETECT = 1
        return ; if Z=0, return and check the next sensor
        goto    DETECTED ;if detected, check US


;;PURPOSE: Checks the ultrasonic, if detects it, it will stop the
motors, rotate the arm
;;      ;and them come back and turn the motors back on so that it can
;;      ;continue to check for barrels again
;;      ;if does not detect, then it will jump to keeping the motors on
and
;;      ;continue to check for barrels again
CHECK_US
        call    ULTRASONIC
        ;C is set when Time_High > 3
        movlw   b'11' ;3
        subwf   Time_High,W
        btfss   STATUS, C
        call    RETRACT_ARM_BACK ;this means C<3, retract arm
        call    MOTOR_ON_RC1
        call    MOTOR_ON_RC2
        goto    CHECK_IRSENSORS   ;this means C>3, go back to checking

DETECTED
        ;at this point, the LSL has been detected, so stop the motors

        call    MOTOR_BOTH_OFF
        ;now check if the LSH detects anything, if it does, it means it
is a
        ;large barrel, if not, then it is a small barrel

        btfss   LS; if the laser detects AND the IR sensors detect, it
is a large barrel
        goto    SHORTBARREL ;it is a short barrel
        goto    TALLBARRELL ;it is a large barel

        ;at this point, done recording and continue with operation

        goto OPERATION_START


END_OPERATION
        ;turn back
        call    RETRACT_ARM

        ;reset the distance
        movlw   b'00110000'
        movwf   Dis_Ones
        movlw   b'00110000'
        movwf   Dis_Tens
        movlw   b'00110000'
        movwf   Dis_Hunds

END_LOOP    ;add in to retrieve final operation time
```

```
        bsf DCA ;turn on motos to travel back
        bsf DCB
        call    CHECK_DISTANCE
        btfss   STATUS, C   ;if set, then end
        goto    END_LOOP ;if not, keep going

        bcf DCA ;turn off DC motors
        bcf DCB
        goto    END_DISPLAY ;exit


END_DISPLAY
        call    Clear_Display
        Display Message3
        call    Switch_Lines
        Display Message4

CHECK_PRESS1
        btfss   PORTB, 1    ;check for input from KEYPAD
        goto    $-1     ;if NOT, keep polling
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        goto    OPTION1

OPTION1     ;checks if * was pressed
        movwf   option_temp
        xorlw   b'00001100'         ; Check to see if 12th key
        btfss   STATUS,Z            ; If status Z goes to 0, it is the
13th key, skip
        goto    OPTION2         ; If not check if it's B
        call    Clear_Display
        goto    Main                ; If it is, restart

OPTION2     ;checks if # was pressed
        movf    option_temp, W
        xorlw   b'00001110'
        btfss   STATUS,Z
        goto    CHECK_PRESS1 ;resume polling
        call    Clear_Display
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    POLL1

POLL1
        btfss   PORTB, 1    ;check for input from KEYPAD
        goto    Polltime1   ;if no input, poll INFO
        swapf   PORTB, W    ;when input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS1 ;check which key was pressed
Polltime1
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
        movlw   "1"     ;displays barrel #
```

```
        call    WR_DATA
        Display Message5
        movfw   barrel1 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel1
        call    Check_Height
        Display Message7
        movfw   barrel1+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel1+2
        call    WR_DATA
        movfw   barrel1+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL1


CHECKPRESS1
BACKWARD1   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD1         ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL7
FORWARD1    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL1 ;resume polling
        call    Clear_Display
        goto    POLL2
;**************************************
; BARREL2
;**************************************
POLL2
        btfss   PORTB, 1    ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime2
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS2

Polltime2
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
```

```
        movlw   "2"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel2 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel2
        call    Check_Height
        Display Message7
        movfw   barrel2+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel2+2
        call    WR_DATA
        movfw   barrel2+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL2

CHECKPRESS2
BACKWARD2   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD2         ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL1
FORWARD2    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL2 ;resume polling
        call    Clear_Display
        goto    POLL3

;****************************************
; BARREL3
;****************************************

POLL3
        btfss   PORTB, 1    ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime3
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS3

Polltime3
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
```

```
        call    Realtime     ;displays Real Time
        Display Message2     ;displays B:
        movlw   "3"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel3 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel3
        call    Check_Height
        Display Message7
        movfw   barrel3+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel3+2
        call    WR_DATA
        movfw   barrel3+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL3
CHECKPRESS3
BACKWARD3   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD3        ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL2
FORWARD3    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL2 ;resume polling
        call    Clear_Display
        goto    POLL4

;****************************************
; BARREL4
;****************************************

POLL4
        btfss   PORTB, 1     ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime4
        swapf   PORTB, W     ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1     ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS4
Polltime4
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
```

```
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
        movlw   "4"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel4 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel4
        call    Check_Height
        Display Message7
        movfw   barrel4+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel4+2
        call    WR_DATA
        movfw   barrel4+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL4
CHECKPRESS4
BACKWARD4   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD4        ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL3
FORWARD4    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL4 ;resume polling
        call    Clear_Display
        goto    POLL5

;***************************************
; BARREL5
;***************************************

POLL5
        btfss   PORTB, 1    ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime5
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS5

Polltime5
        movlw   "T"     ;displays T for Real Time
```

```
        call    WR_DATA
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
        movlw   "5"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel5 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel5
        call    Check_Height
        Display Message7
        movfw   barrel5+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel5+2
        call    WR_DATA
        movfw   barrel5+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL5

CHECKPRESS5
BACKWARD5   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD5        ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL4
FORWARD5    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL5 ;resume polling
        call    Clear_Display
        goto    POLL6


;****************************************
; BARREL6
;****************************************

POLL6
        btfss   PORTB, 1    ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime6
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS6
```

```
Polltime6
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
        movlw   "6"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel6 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel6
        call    Check_Height
        Display Message7
        movfw   barrel6+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel6+2
        call    WR_DATA
        movfw   barrel6+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL6
CHECKPRESS6
BACKWARD6   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD6        ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL5
FORWARD6    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL6 ;resume polling
        call    Clear_Display
        goto    POLL7

;***************************************
; BARREL7
;***************************************

POLL7
        btfss   PORTB, 1    ;check for input from KEYPAD
        ;goto   $-1     ;if NOT, keep polling
        goto    Polltime7
        swapf   PORTB, W    ;When input is detected, read it in to W
        andlw   0x0F
        btfsc   PORTB, 1    ;keep iterating until key is released
        goto $-1
        goto    CHECKPRESS7
```

```
Polltime7
        movlw   "T"     ;displays T for Real Time
        call    WR_DATA
        call    Realtime    ;displays Real Time
        Display Message2    ;displays B:
        movlw   "7"     ;displays barrel #
        call    WR_DATA
        Display Message5
        movfw   barrel7 ;T/S/E/HF/F
        call    Check_Type
        call    Switch_Lines
        Display Message6
        movfw   barrel7
        call    Check_Height
        Display Message7
        movfw   barrel7+3 ;ten digit first, how is this stored? Leave
as O's for now
        call    WR_DATA
        movfw   barrel7+2
        call    WR_DATA
        movfw   barrel7+1
        call    WR_DATA
        Display Message8
        call    HalfS
        call    Clear_Display
        goto    POLL7
CHECKPRESS7
BACKWARD7   ;checks if 1 was pressed
        movwf   option_temp
        xorlw   b'00000000'         ;checks to see if "1" was pressed
        btfss   STATUS,Z            ;if status Z goes to 0, it is not
"1"
        goto    FORWARD7         ;if not, check to see if "2" was
pressed
        call    Clear_Display
        goto    POLL6
FORWARD7    ;checks if 2 was pressed
        movf    option_temp, W
        xorlw   b'00000001'
        btfss   STATUS,Z
        goto    POLL7 ;resume polling
        call    Clear_Display
        goto    POLL1

        goto    $
        ;1: Stores Tall/Short Barrel, Stores E/HF/F
        ;2: Location (stores distance < 256 cm)
        ;3: Location (if distance > 256 cm)
;;
;;
;;


;;ShiftDisplayLeft
;       ;call       Clear_Display
;
;;      Display     Welcome_Msg2
```

```
;;ChangeToQuestionMark
;;      movlw       b'11001011'
;;      call        WR_INS
;;      movlw       "?"
;;      call        WR_DATA
;
;
;
;;Left  movlw       b'00011000'     ;Move to the left
;;      call        WR_INS
;;      call        HalfS
;;      goto        Left            ;repeat operation
;;
;***************************************
; MAIN PROGRAM SUBROUTINES
;***************************************


;***************************************
; IR SENSOR CODE
;***************************************


CHECK_IR1
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11000001' ;to select IR1
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR2
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11001001' ;to select IR2
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR3
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11010001' ;to select IR3
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR4
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT
```

```
        movlw   b'11011001' ;to select IR4
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR5

        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11100001' ;to select IR5
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR6
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11101001' ;to select IR6
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR7
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11110001' ;to select IR7
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
CHECK_IR8
        ;initialize IR_DETECT
        movlw   b'0'
        movwf   IR_DETECT

        movlw   b'11111001' ;to select IR8
        movwf   ad_store
        movfw   ad_store
        call    IR_MAINLOOP
        return ;GO BACK TO OPERATION CODE
IR_MAINLOOP
        movfw   ad_store ;storing the ADCON value
        call    AD_CONV
        movwf   voltage_IR
        call    CHECK_IR
        btfss   STATUS,C
        goto    DISPLAYCHECKHIGH
        goto    DISPLAYLOW
        return
        ;C is set when voltage >= 4.1
```

```
CHECK_IR
        movlw   b'10111101'
        subwf   voltage_IR,W
        return

DISPLAYLOW

        movlw   d'0'
        movwf   lastop_IR
        movlw   b'0'
        movwf   IR_DETECT
        return

DISPLAYCHECKHIGH
        movfw   lastop_IR ;check if last call was a 1
        xorlw   d'1'
        btfsc   STATUS, Z
        goto INCREMENT_IR ;if it is, increment
        movlw   d'1' ;else, set lastop = 1
        movwf   lastop_IR
        movlw   d'1' ;set counter = 1
        movwf   counter_IR
        goto    IR_MAINLOOP
        ;return

INCREMENT_IR
        movfw   counter_IR
        xorlw   d'4'
        btfsc   STATUS,Z
        goto    DISPLAYHIGH
        incf    counter_IR
        movfw   counter_IR
        xorlw   d'4'
        btfsc   STATUS,Z
        goto    DISPLAYHIGH
        goto    IR_MAINLOOP
        ;return
DISPLAYHIGH

        movlw   b'1'
        movwf   IR_DETECT
        return

;***************************************
; ULTRASONIC SENSOR CODE
;***************************************


ULTRASONIC
        movlw   d'16' ;initialize timer module
        movwf   T1CON
        movlw   d'0'
        movwf   TMR1L
        movlw   d'0'
        movwf   TMR1H

        bsf US_TRIG ;10us TRIGGER HIGH
```

```
        call    DelayL
        bcf US_TRIG ;TRIGGER LOW

        btfss   US_ECHO ;waiting to detect echo (HIGH)
        goto    $-1
        bsf T1CON, 0 ;turn timer on / TMR1ON=1
        btfsc   US_ECHO ;waiting for echo to go LOW
        goto    $-1
        bcf T1CON, 0 ;turn timer off

        movfw   TMR1L
        movwf   Time_Low
        movfw   TMR1H
        movwf   Time_High

        return

;;***************************************
;; ENCODER SUBROUTINE
;;***************************************
;;
;;      btfss   PORTB, 1
;;      goto    $-1
;;      call    Clear_Display
;;      call    Distance_Count
;;      movfw   Dis_Hunds
;;      call    WR_DATA
;;      movfw   Dis_Tens
;;      call    WR_DATA
;;      movfw   Dis_Ones
;;      call    WR_DATA
;;      btfsc   PORTB, 1
;;      goto    $-1
;;      goto    TEST_ENCODER
;;      goto    $
;
;****************************************
; DISTANCE COUNT SUBROUTINE
;****************************************
Distance_Count
        ;movlw      0x0C    ;Wait to begin
        ;Keypad

        movfw       Dis_Ones
        xorlw       b'00111001'
        btfsc       STATUS,Z
        goto        Skip_Ten
        incf        Dis_Ones,1
        return

Skip_Ten    movfw       Dis_Tens
        xorlw       b'00111001'
        btfsc       STATUS,Z
        goto        Skip_Hund
        incf        Dis_Tens,1
        movlw       b'00110000'
        movwf       Dis_Ones
```

```
        return

Skip_Hund   movfw       Dis_Hunds
        xorlw       b'00111001' ;ASCII 9
        btfsc       STATUS,Z
        goto        Skip_Thou
        incf        Dis_Hunds,1
        movlw       b'00110000'
        movwf       Dis_Ones
        movwf       Dis_Tens
        return

Skip_Thou   incf        Dis_Thous,1
        movlw       b'00110000' ;ASCII 0
        movwf       Dis_Ones
        movwf       Dis_Tens
        movwf       Dis_Hunds
        return

;;***************************************
;; RETRACT ARM SUBROUTINE
;;***************************************
;
;RETRACT_ARM_BACK
;       ;bcf    DCA1 ;turn off wheel motors
;       ;bcf    DCB1 ;turn off wheel motors
;       bcf DCC1
;       bsf DCC2 ;turn on DC motor
;       ;for X seconds, need to be tested
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;
;       bcf DCC2 ;turn off DC motor
;
;
;       ;dont move for 3 seconds
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;
;       bsf DCC1
;       bcf DCC2 ;reverse arm DC direction
;
;
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;
;       bcf DCC1
;
;
```

```
;       return


;*TEST RETRACT_ARM FOR DEMO*

;RETRACT_ARM_BACK
;       bcf DCC1
;       bsf DCC2 ;turn on DC motor
;
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;
;
;       bsf DCC1
;       bcf DCC2 ;reverse arm DC direction
;
;
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;       call    HalfS
;
;
;
;       return
;

;***************************************
; CHECK DISTANCE SUBROUTINE
;***************************************

CHECK_DISTANCE

        movfw   Dis_Hunds ;this must be <=4
        movlw   b'00110100' ;max value for the hundreds place
        subwf   Dis_Hunds, W ;Dis_Hunds <- Dis_Hunds - 4
        return
;***************************************
; SHORTBARREL SUBROUTINE
;***************************************


SHORTBARREL
```

```
        incf    barrelnum, 1 ;increment barrel count

        call    CHECK_IR1 ;check if IR1 detects
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    SHORTBARREL1  ;IR1 does not detect, then check IR5
        goto    SFULLORHALF ;IR1 detects, at this point, the barrel is
either FULL or HALFFULL

SHORTBARREL1
        call    CHECK_IR5 ;check if IR5 detects
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    RECORD_SE ;IR1 and IR5 both don't detect, it is
SMALL+EMPTY
        goto    SFULLORHALF  ;IR5 detects, at this point, the barrel is
either FULL or HALFFULL


SFULLORHALF
        call    CHECK_IR3 ;check if IR3 detects
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    SFULLORHALF1 ;IR3 does not detect, check IR7 on the
other side
        goto    RECORD_SF   ;IR3 detects, it must be SMALL + FULL
SFULLORHALF1
        call    CHECK_IR7 ;check if IR7 detects
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    RECORD_SHF ;IR3 and IR7 does not detect, but either IR1
or IR5 detected, so this must be SMALL + HALFFULL
        goto    RECORD_SF  ;IR7 detects, it must be SMALL + FULL


;****************************************
; TALLBARREL SUBROUTINE
;****************************************
TALLBARRELL
        incf    barrelnum, 1

        call    CHECK_IR2 ;check if IR2 detects
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    TALLBARRELL1 ;if IR2 does not detect, check IR6 on the
other side of the arm
        goto    TFULLORHALF  ;if IR2 detects, the barrel is either FULL
or HALFFULL
TALLBARRELL1
        call    CHECK_IR6
        movfw   IR_DETECT
        xorlw   b'1'
```

```
        btfss   STATUS, Z
        goto    RECORD_TE ;if IR6 does not detect either, it must be
EMPTY
        goto    TFULLORHALF ;if IR6 detects, the barrel is either FULL
or HALFFULL

TFULLORHALF
        call    CHECK_IR4
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    TFULLORHALF1 ;if IR4 does not detect, check IR8 on the
other side of the arm
        goto    RECORD_TF   ;if IR4 detects, it must be TALL + FULL
TFULLORHALF1
        call    CHECK_IR8
        movfw   IR_DETECT
        xorlw   b'1'
        btfss   STATUS, Z
        goto    RECORD_THF  ;if IR8 does not detect either, it must be
TALL + HALLFULL
        goto    RECORD_TF ;if IR8 detects, it is TALL + FULL

;***************************************
; MOVE INFO TO BARREL_DATA SUBROUTINE
;***************************************

RECORD_SE
        movlw   b'00001100' ;(S=1/T=0)/E/HF/F
        movwf   barrel_data ;store it so it can be recorded
        goto    RECORD
RECORD_SHF
        movlw   b'00001010'
        movwf   barrel_data
        goto    RECORD
RECORD_SF
        movlw   b'00001001'
        movwf   barrel_data
        goto    RECORD

RECORD_TE
        movlw   b'00000100'
        movwf   barrel_data
        goto    RECORD
RECORD_THF
        movlw   b'00000010'
        movwf   barrel_data
        goto    RECORD
RECORD_TF
        movlw   b'00000001'
        movwf   barrel_data
        goto    RECORD

;***************************************
; MAIN RECORD
;***************************************
```

```
RECORD      ;main record
        ;first need to determine which barrel it is

B_ONE
        movfw   barrelnum ;move barrelnum to working register
        xorlw   b'00000001' ; 1
        btfsc   STATUS, Z
        goto    RECORD_ONE
        goto    B_TWO

B_TWO
        movfw   barrelnum
        xorlw   b'00000010' ; 2
        btfsc   STATUS, Z
        goto    RECORD_TWO
        goto    B_THREE

B_THREE
        movfw   barrelnum
        xorlw   b'00000011' ;3
        btfsc   STATUS, Z
        goto    RECORD_THREE
        goto    B_FOUR

B_FOUR
        movfw   barrelnum
        xorlw   b'00000100' ;4
        btfsc   STATUS, Z
        goto    RECORD_FOUR
        goto    B_FIVE

B_FIVE
        movfw   barrelnum
        xorlw   b'00000101' ;5
        btfsc   STATUS, Z
        goto    RECORD_FIVE
        goto    B_SIX

B_SIX
        movfw   barrelnum
        xorlw   b'00000110' ;6
        btfsc   STATUS, Z
        goto    RECORD_SIX
        goto    B_SEVEN

B_SEVEN
        goto    RECORD_SEVEN ;has to be barrel 7 at this point


;***************************************
; RECORD WHEN BARREL NUMBER IS KNOWN
;***************************************


RECORD_ONE
        ;stores the E/HF/F bits
```

```
        movfw   barrel_data
        movwf   barrel1 ;move the data into barrel1, althought only the
last three move bits are important
        goto    OPERATION_START ;go back to program

RECORD_TWO
        movfw   barrel_data
        movwf   barrel2
        goto    OPERATION_START

RECORD_THREE
        movfw   barrel_data
        movwf   barrel3
        goto    OPERATION_START

RECORD_FOUR
        movfw   barrel_data
        movwf   barrel4
        goto    OPERATION_START

RECORD_FIVE
        movfw   barrel_data
        movwf   barrel5
        goto    OPERATION_START

RECORD_SIX
        movfw   barrel_data
        movwf   barrel6
        goto    OPERATION_START

RECORD_SEVEN
        movfw   barrel_data
        movwf   barrel7
        goto    OPERATION_START

;***************************************
; ULTRASONIC DELAYS (10 us)
;***************************************

DelayL
        movlw   0x30        ; b'00110000'
        movwf   0x53        ; general purpose register
CONT3L
        decfsz  0x53, f
        goto    CONT3L
        return

;***************************************
; MOTOR SUBROUTINE (PWM)
;***************************************

MOTOR_ON_RC1
        movlw   b'11111111'
        movwf   CCPR2L
        bsf PORTC, 1
        return
MOTOR_ON_RC2
```

```
        movlw   b'11111111'
        movwf   CCPR1L

        bsf PORTC,2
        return
MOTOR_BOTH_OFF
        movlw   b'00000000'
        movwf   CCPR2L
        movwf   CCPR1L
        return


    ;100% 11111111 255
    ;80%    11000111 199
    ;60%    10010101 149
    ;0%     00000000
;***************************************
; LCD control
;***************************************
Switch_Lines
        movlw   B'11000000'
        call    WR_INS
        return

Clear_Display
        movlw   B'00000001'
        call    WR_INS
        return

;***************************************
; Delay 0.5s
;***************************************
HalfS
    local   HalfS_0
      movlw 0x88
      movwf COUNTH
      movlw 0xBD
      movwf COUNTM
      movlw 0x03
      movwf COUNTL

HalfS_0
      decfsz COUNTH, f
      goto   $+2
      decfsz COUNTM, f
      goto   $+2
      decfsz COUNTL, f
      goto   HalfS_0

      goto $+1
      nop
      nop
        return
;***************************************
; STORING BARREL INFO ON LCD SUBROUTINE
;***************************************
```

```
Check_Type
CHECKE
        movwf   barreltemp ;STORE IT TEMPORARY, LEST XORLW WILL ALTER
IT

        ;check if barrel has been accessed
        movfw   barreltemp
        xorlw   b'01011000' ;ASCII X
        btfsc   STATUS, Z ;Z=1 if ASCII X
        goto    PRINTDEFAULT ;Z=1, so display X
        movfw   barreltemp ;Z!=1, so continue
        btfss   barreltemp,2
        goto    CHECKHF
        movlw   "E"
        call    WR_DATA
        return

CHECKHF
        movfw   barreltemp
        btfss   barreltemp,1
        goto    CHECKF
        movlw   "H"
        call    WR_DATA
        movlw   "F"
        call    WR_DATA
        return

CHECKF
        ;must be FULL at this point
        movlw   "F"
        call    WR_DATA
        return
PRINTDEFAULT
        movlw   "X"
        call    WR_DATA
        return

Check_Height
CHECKSHORT

        movwf   barreltemp ;STORE IN HERE TEMPORARILY

        ;check if barrel has been accessed
        movfw   barreltemp
        xorlw   b'01011000' ;ASCII X
        btfsc   STATUS, Z ;Z=1 if ASCII X
        goto    PRINTDEFAULT1 ;Z=1, so display X
        movfw   barreltemp ;Z!=1, so continue
        btfss   barreltemp,3
        goto    CHECKTALL
        movlw   "S"
        call    WR_DATA
        movlw   " "
        call    WR_DATA
        return
CHECKTALL   ;must be TALL at this point
```

```
        movlw   "T"
        call    WR_DATA
        movlw   " "
        call    WR_DATA
        return

PRINTDEFAULT1
        movlw   "X"
        call    WR_DATA
        movlw   " "
        call    WR_DATA
        return
;******* LCD-related subroutines *******


    ;***********************************
InitLCD
    bcf STATUS,RP0
    bsf E     ;E default high

    ;Wait for LCD POR to finish (~15ms)
    call lcdLongDelay
    call lcdLongDelay
    call lcdLongDelay

    ;Ensure 8-bit mode first (no way to immediately guarantee 4-bit
mode)
    ; -> Send b'0011' 3 times
    movlw   b'00110011'
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay
    movlw   b'00110010'
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay

    ; 4 bits, 2 lines, 5x7 dots
    movlw   b'00101000'
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay

    ; display on/off
    movlw   b'00001100'
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay

    ; Entry mode
    movlw   b'00000110'
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay

    ; Clear ram
    movlw   b'00000001'
```

```
    call    WR_INS
    call lcdLongDelay
    call lcdLongDelay
    return
    ;************************************

    ;ClrLCD: Clear the LCD display
ClrLCD
    movlw   B'00000001'
    call    WR_INS
    return

    ;****************************************
    ; Write command to LCD - Input : W , output : -
    ;****************************************
WR_INS
    bcf     RS              ;clear RS
    movwf   com             ;W --> com
    andlw   0xF0            ;mask 4 bits MSB w = X0
    movwf   PORTD           ;Send 4 bits MSB
    bsf     E               ;
    call    lcdLongDelay    ;__    __
    bcf     E               ;  |__|
    swapf   com,w
    andlw   0xF0            ;1111 0010
    movwf   PORTD           ;send 4 bits LSB
    bsf     E               ;
    call    lcdLongDelay    ;__    __
    bcf     E               ;  |__|
    call    lcdLongDelay
    return

    ;****************************************
    ; Write data to LCD - Input : W , output : -
    ;****************************************
WR_DATA
    bsf     RS
    movwf   dat
    movf    dat,w
    andlw   0xF0
    addlw   4
    movwf   PORTD
    bsf     E               ;
    call    lcdLongDelay    ;__    __
    bcf     E               ;  |__|
    swapf   dat,w
    andlw   0xF0
    addlw   4
    movwf   PORTD
    bsf     E               ;
    call    lcdLongDelay    ;__    __
    bcf     E               ;  |__|
    return

lcdLongDelay
    movlw d'20'
    movwf lcd_d2
```

```
LLD_LOOP
    LCD_DELAY
    decfsz lcd_d2,f
    goto LLD_LOOP
    return
;*********
; BIN2BCD
; Converts a binary number to ASCII
; characters for display on the LCD
; Written by: A. Borowski
; Sourced from: piclist.com --> 8 bit to ASCII Decimal 3 digits
;**********
BIN2BCD
    movlw 8
    movwf count
    clrf huns
    clrf tens
    clrf ones

BCDADD3

    movlw 5
    subwf huns, 0
    btfsc STATUS, C
    CALL ADD3HUNS

    movlw 5
    subwf tens, 0
    btfsc STATUS, C
    CALL ADD3TENS

    movlw 5
    subwf ones, 0
    btfsc STATUS, C
    CALL ADD3ONES

    decf count, 1
    bcf STATUS, C
    rlf binary_num, 1
    rlf ones, 1
    btfsc ones,4 ;
    CALL CARRYONES
    rlf tens, 1

    btfsc tens,4 ;
    CALL CARRYTENS
    rlf huns,1
    bcf STATUS, C

    movf count, 0
    btfss STATUS, Z
    goto BCDADD3

    movf huns, 0 ; add ASCII Offset
    addlw h'30'
    movwf huns
```

```
    movf tens, 0 ; add ASCII Offset
    addlw h'30'
    movwf tens

    movf ones, 0 ; add ASCII Offset
    addlw h'30'
    movwf ones
    return

ADD3HUNS
    movlw 3
    addwf huns,1
    return

ADD3TENS
    movlw 3
    addwf tens,1
    return

ADD3ONES
    movlw 3
    addwf ones,1
    return




CARRYONES
    bcf ones, 4
    bsf STATUS, C
    return

CARRYTENS
    bcf tens, 4
    bsf STATUS, C
    return

     ;call  AD_CONV
     ;call  WR_DATA
     ;call  HalfS
     ;call  Clear_Display
     ;goto  Main

;*********
; ADC
;**********
    goto    INITA
INITA   bsf STATUS,RP0 ;select bank 1
    bcf INTCON,GIE ;disable global interrupt
    movlw   B'00000000' ;configure ADCON1
    movwf   ADCON1
    clrf    TRISB ;configure PORTB as output
    bcf STATUS,RP0 ;select bank 0
    goto    ADSTART
;***************************************************************
; MAIN PROGRAM
```

```
;****************************************************************
ADSTART call    AD_CONV ;call the A2D subroutine
    movwf   PORTB ;display the high 8-bit result to the LEDs
ENDLP   goto    ENDLP ;endless loop
;****************************************************************
; AD CONVERT ROUTINE
;****************************************************************
AD_CONV ;movlw  B'10000001' ;configure ADCON0
    movwf   ADCON0
    call    TIM20 ;wait for required acquisition time
    bsf ADCON0,GO ;start the conversion
WAIT    btfsc   ADCON0,GO ;wait until the conversion is completed
    goto    WAIT ;poll the GO bit in ADCON0
    movf    ADRESH,W ;move the high 8-bit to W
    return
;***************************************************************
; TIME DELAY ROUTINE FOR 20us
;
; - delay of 400 cycles
; - 400*0.05us = 20us
;***************************************************************
TIM20   movlw   084H ;1 cycle
    movwf   TIMCNT ;1 cycle
TIMLP   decfsz  TIMCNT,F ;(3*132)-1 = 395 cycles
    goto    TIMLP
    nop ;1 cycle
    return  ;2 cycles

;***************************************
; Real Time
;***************************************

show_RTC
        ;clear LCD screen
        movlw   b'00000001'
        call    WR_INS

        ;Get year
        ;movlw  "2"             ;First line shows 20**/**/**
        ;call   WR_DATA
        ;movlw  "0"
        ;call   WR_DATA
        ;rtc_read   0x06        ;Read Address 0x06 from DS1307---year
        ;movfw  0x77
        ;call   WR_DATA
        ;movfw  0x78
        ;call   WR_DATA

        ;movlw  "/"
        ;call   WR_DATA

        ;Get month
        ;rtc_read   0x05        ;Read Address 0x05 from DS1307---month
        ;movfw  0x77
        ;call   WR_DATA
        ;movfw  0x78
        ;call   WR_DATA
```

```
        ;movlw  "/"
        ;call   WR_DATA

        ;Get day
        ;rtc_read   0x04        ;Read Address 0x04 from DS1307---day
        ;movfw  0x77
        ;call   WR_DATA
        ;movfw  0x78
        ;call   WR_DATA

        ;movlw  B'11000000'     ;Next line displays (hour):(min):(sec)
**:**:**
        ;call   WR_INS          ;NEXT LINE

        ;Get hour
        ;rtc_read   0x02        ;Read Address 0x02 from DS1307---hour
        ;movfw  0x77
        ;call   WR_DATA
        ;movfw  0x78
        ;call   WR_DATA
        ;movlw          ":"
        ;call   WR_DATA



        ;Get minute
Realtime
        rtc_read    0x01        ;Read Address 0x01 from DS1307---min
        movfw   0x77
        call    WR_DATA
        movfw   0x78
        call    WR_DATA
        movlw           ":"
        call    WR_DATA

        ;Get seconds
        rtc_read    0x00        ;Read Address 0x00 from DS1307---
seconds
        movfw   0x77
        call    WR_DATA
        movfw   0x78
        call    WR_DATA
        return

        call    OneS            ;Delay for exactly one seconds and read
DS1307 again
        goto    show_RTC

Operationtime
        rtc_read    0x01        ;Read Address 0x01 from DS1307---min
        movfw   0x77
        movwf   min
        ;call   WR_DATA
        movfw   0x78
        movwf   min+1
        ;call   WR_DATA
```

```
        ;movlw          ":"
        ;call   WR_DATA

        ;Get seconds
        rtc_read    0x00        ;Read Address 0x00 from DS1307---
seconds
        movfw   0x77
        movwf   sec
        ;call   WR_DATA
        movfw   0x78
        movwf   sec+1
        ;call   WR_DATA
        return

        ;call   OneS            ;Delay for exactly one seconds and read
DS1307 again
        ;goto   show_RTC



;;*************************************
;; Setup RTC with time defined by user
;;*************************************
set_rtc_time

        ;rtc_resetAll   ;reset rtc

        ;rtc_set    0x00,   B'10000000'

        ;set time
        ;rtc_set    0x06,   B'00010000'     ; Year
        ;rtc_set    0x05,   B'00000100'     ; Month
        ;rtc_set    0x04,   B'00000110'     ; Date
        ;rtc_set    0x03,   B'00000010'     ; Day
        ;rtc_set    0x02,   B'00010010'     ; Hours
        ;rtc_set    0x01,   B'00110000'     ; Minutes
        ;rtc_set    0x00,   B'00000000'     ; Seconds
        ;return

;*************************************
; Delay 1s
;*************************************
OneS
        local   OneS_0
      movlw 0x10
      movwf COUNTH1
      movlw 0x7A
      movwf COUNTM1
      movlw 0x06
      movwf COUNTL1

OneS_0
      decfsz COUNTH1, f
      goto   $+2
      decfsz COUNTM1, f
      goto   $+2
      decfsz COUNTL1, f
```

```
  goto   OneS_0

  goto $+1
  nop
  nop
    return


END
```