**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**JNANA SANGAMA, BELAGAVI-590018**

****

**2020-21**

**A Project Report of Phase-II On**

**“FIRE FIGHTING ROBOT”**

**Submitted in partial fulfillment of the requirements for the award of degree**

BACHELOR OF ENGINEERING

IN

MECHANICAL ENGINEERING

BY

**VIKAS N 1GC17ME056**

**DARSHAN B 1GC15ME016**

**SYED SAHIL 1GC16ME083**

**RAKSHITH R 1GC16ME061**

**Under the esteemed guidance of**

**Dr. MOHAMMED AQUIL AHMED**

****

**Department of Mechanical Engineering**

**GHOUSIA COLLEGE OF ENGINEERING**

**RAMANAGARAM-562159**

**2020-2021**

**GHOUSIA COLLEGE OF ENGINEERING**

**Department of Mechanical Engineering**

**RAMANAGARAM - 562 159**

****

**CERTIFICATE**

Certified that the Project work of Phase-II entitled **“FIRE FIGHTING ROBOT”**

carried out by Mr. **VIKAS N (1GC17ME056), DARSHAN B (1GC15ME016),**

**SYED SAHIL(1GC16ME083), RAKSHITH R (1GC16ME061)**

a bonafied students of VII semester Mechanical Engineering in

partial fulfillment for the award of Bachelor of Engineering in Ghousia college of engineering

of Visvesvaraya Technological University, Belagavi during the

year of 2020-21. It is certified that all correction/suggestions indicated for Internal

Assessment have been incorporated in the report.

The project report has been approved as it satisfied the academic requirements in respect

of project work phase-II prescribed for the side degree.

**----------------------------------- -------------------------------**

**Dr. MOHAMMED AQUIL AHMED Dr. MOHAMMED MOHSIN ALI H**

**Assistant Professor, Professor & HOD,**

**Dept of ME, GCE Dept of ME, GCE**

**Ramanagaram. Ramanagaram**

**Acknowledgement**

It is our proud privilege and duty to acknowledge the kind of help and guidance

received from several people in preparation of this report. It would not have been possible

to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost, we wish to record my sincere gratitude to the Management

Ghousia College of Engineering and to our beloved Principal, **Dr. Mohammed Zahid Ansari,**

Principal, **Ghousia College of Engineering, Ramanagarm** for his constant support and

encouragement in preparation of this report and for making available library and laboratory

facilities needed to prepare this report.

Our sincere thanks to **Dr. Mohammed Mohsin Ali H**, Head of the Department of Mechanical Engineering, Ghousia of College Engineering, Ramanagarm for his valuable suggestions and guidance throughout the period of this report.

We express our sincere gratitude to our guide, **Dr. Mohammed Aquil Ahmed,** Assistant professor, Department of Mechanical Engineering, GCE, Ramanagarm for guiding us in investigations for this seminar and in carrying out experimental work. The numerous discussions with him were extremely helpful. We hold him in esteem for guidance, encouragement and

inspiration received from him.

The seminar on **PROJECT PHASE II** of **FIRE FIGHTING ROBOT** was very helpful to us in

giving the necessary background information and inspiration in choosing this topic for the

seminar. Our sincere thanks to **Dr. Mohammed Aquil Ahmed** Seminar Coordinator for having supported the work related to this project. His contributions and technical support

in preparing this report are greatly acknowledged.

Last but not the least, we wish to thank our parents for financing our studies in this

college as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

Place: Ramanagara B. Darshan. (1GC15ME016)

Vikas.N (1GC17ME056)

Rakshith.R (1GC16ME061)

Syed Sahil (1GC16ME083)

i

**Table of Contents**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1.** | **Introduction** | | |  | **1** |
| **2.** | **Problem description** | | | | **2** |
| **3.** | **Project Specifications** | | | | **3** |
|  | 3.1 | Overall design | |  | 3 |
|  | 3.2 | Task assignment | | | 9 |
|  | 3.3 | Initial parts list | |  | 10 |
| **4. Systems design and integration** | | | | | **11** |
|  | 4.1 | Master controller | | | 11 |
|  |  | 4.1.1 | Hardware | | 11 |
|  |  | 4.1.2 | Software | | 11 |
|  |  |  | 4.1.2.1 | RTOS | 12 |
|  |  |  | 4.1.2.2 | Core software | 13 |
|  |  |  | 4.1.2.3 | Flame tracking hysteresis | 14 |
|  |  |  | 4.1.2.4 | Right angle turning | 14 |
|  | 4.2 | Motor control | |  | 14 |
|  |  | 4.2.1 | Decision on parts | | 14 |
|  |  | 4.2.2 | Testing process | | 15 |
|  |  | 4.2.3 | Hardware | | 16 |
|  |  | 4.2.4 | Software | | 17 |
|  | 4.3 | Proximity control | | | 18 |
|  |  | 4.3.1 | Decision on parts | | 18 |
|  |  | 4.3.2 | Testing process | | 18 |
|  |  | 4.3.3 | Hardware | | 20 |
|  |  | 4.3.4 | Software | | 21 |
|  | 4.4 | Fire detection control | | | 22 |
|  |  | 4.4.1 | Decision on parts | | 22 |
|  |  | 4.4.2 | Testing process | | 23 |
|  |  |  | 4.4.2.1 | Cover design | 24 |
|  |  | 4.4.3 | Hardware | | 26 |
|  |  |  | 4.4.3.1 | Sensor terminal | 26 |
|  |  |  | 4.4.3.2 | Circuit design | 29 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 4.4.4 | Software |  | 30 |
|  |  | 4.4.4.1 | Pseudo code | 30 |
|  |  | 4.4.4.2 | Testing of software | 30 |
|  | 4.5 | Fan controller |  | 30 |
|  | 4.6 | Utility board |  | 31 |
|  |  | 4.6.1 PCB Manufacturing | | 32 |
|  | 4.7 | Chassis  Design |  | 33 |
|  | 4.8 | Integration step | | 36 |
| **5.** | **Evaluation** | |  | **38** |
| **6.** | **Conclusion** | |  | **39** |
| **7.** | **References** | |  | **40** |
| **Appendix A:** Master controller code | | | | 41 |
| **Appendix B:** Fire detection controller code | | | | 50 |
| **Appendix C:** Motor controller code | | | | 53 |
| **Appendix D:** Proximity code | | | | 55 |

## 1. Introduction

This report illustrates the design and implementation of our team’s firefighting robot for the Industrial Systems Design and Integration paper. The robot must run automatically, avoiding obstacles and at the same time find and track two flames (candle light) and extinguish them. To achieve the best performance with an effective implementation, we have taken a modular design strategy, where the robot is divided into a number of logical modules based on functionality.

Our design consists of four main modules:

1. Master controller
2. Motor control
3. Proximity control
4. Fire detection

Each module is associated with appropriate sensors and a microcontroller, which is then interfaced to the master controller.

In the next two chapters, the specification of the project is identified. Then, each module’s design and implementation is discussed in their own sections. Section 4.8 describes the integration steps. And finally, the report concludes with an evaluation of our project.

**2. Problem Description**

The goals of this project are listed below:

* It must run automatically
* It must avoid any obstacles present

* It must track and find flames (candle lights) and extinguish them without making direct contact
* The entire project must not exceed a budget of NZ$500 ($100 per person)

### 

**3. Project Specifications**

**3.1 Overall Design**

After some discussions within the team and with some analysis of previous robots, we decided on a robot with the following design:

* Using 6 panels of sensors, capable of detecting flame sources in a 360 degrees fashion. A lot of the older projects seem sluggish, in that they have to stop, spin around to find the flame,

and continue on in that direction for a small amount of time, stop and spin around again

* Use Servos or DC motors
* Use IR sensors for proximity detection

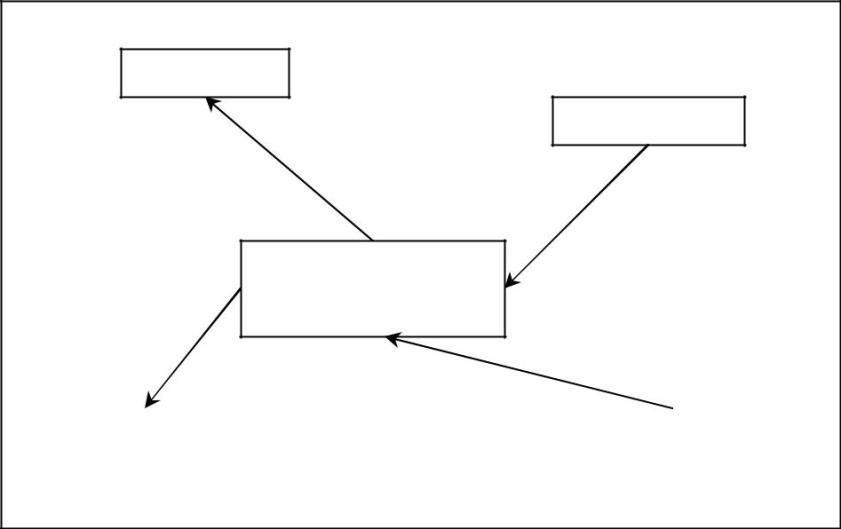
The project also has a modular design as illustrated in figure 3.1, where all the tasks to be performed by our robot are split into:

* *Proximity*, this involves the interfacing of a number of proximity sensors to detect thepresence of objects

* *Motor*, this involves the making of a motor controller that can interpret commands sent to it,and perform corresponding motor actions

* *Flame Tracking*, this involves the interfacing of sensors in such a way that we can reliablydetect flame sources in a 360 degrees fashion
* *Firefighting*, this involves finding a suitable way to put out a fire

* *Core unit*, a core unit will be present to co-ordinate the actions of all the sub units listed above



Motor

Proximity

Master controller

(

Core unit

)

Fire

fighting

Flame tracking

Figure 3.1: Modular design of the robot

The modular design of our robot provides a number of advantages:

* We have a clear separation of tasks, and thus each team member is more likely to be able to meet the deadline

* Having all components required for a particular set of sensors on a single board means that each team member can work independently, according to their own schedules

* Because each board is independent, they can be tested independently before the integration step

* Functionalities of the finished robot can be switched on and off by simply removing power from the individual modules. This makes debugging the final robot easier

* Modular designs have many redundancies, and as such as less prone to component failures. Broken parts can be swapped out very quickly

Aside from the modular design strategy, we also placed some basic constraints on our robot. These constraints serve as a design guideline whenever a team member is in doubt as to what to implement:

* All modules must conform to the specifications given. As long as the specifications are met any implementation is acceptable

* The robot will not stop moving under any circumstances. Many previous robots suffer from sluggish movements, as they often start and stop, and have significant delays between actions. We have decided from the beginning that our robot will be very responsive, and to reflect this decision, the stop and reverse command are never built into the motor controller specifications (refer to section 4.2)

* Rather than expensive components, cheaper ones are preferred. This way we are much more flexibility should a component fail or if the specification changes

With constraints and modules determined for the robot, we produced the following specifications for the different modules.

Motor control

* Reaction speed lower than 25ms
* Must have speed control
* Must have special modes (turn 180° etc)
* Must be based on interrupt
* Power requirement must be below 9V batteries

System input

* Number of pins: 5
* Pin-in

Po-External Interrupt

P1-Mo

P2-M1

P3-M2

P4-M3

Figure 3.2: Motor control specification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SPECIAL**  **M3** | **L**  **M2** | **R**  **M1** | **SPEED**  **Mo** | **EVENT** |
| 0 | 0 | 0 | 0 | STOP |
| 1 | 1 | 1 | 1 | TURN 180® |
| 0 | 1 | 0 | 0 | TURN LEFT, SLOW |
| 0 | 1 | 0 | 1 | TURN LEFT, FAST |
| 0 | 0 | 1 | 0 | TURN RIGHT, SLOW |
| 0 | 0 | 1 | 1 | TURN RIGHT, FAST |
| 0 | 1 | 1 | 0 | FORWARD, SLOW |
| 0 | 1 | 1 | 1 | FORWARD, FAST |
| 1 | 0 | 0 | 0 | BACKWORD, SLOW |
| 1 | 0 | 0 | 1 | BACKWORD, FAST |

Input pattern

Input sequence

* M3-Mo written
* Interrupt (on Po) is triggered

Figure 3.3: Motor control input summary

Collision Detection

* Sensor to be placed wide apart to cover the entire front cross section.
* Calibration graph (average result over st tests)
* Report distance should not be too far.
* Sensor must be fast (Faster than 100 ms update internals if possible)
* Compact board design.
* Minimum noise.
* Power requirement must be below 9V batteries.

System output

* Number of pins: 4
* Pinout

Po-Read inhibit

P1-Mo

P2-M1

P3-Reserved, Leave on low.

Output pattern

|  |  |  |  |
| --- | --- | --- | --- |
| RESERVED  M2 | L  M1 | R  Mo | EVENT |
| 0 | 0 | 0 | N0 OBJECT |
| 0 | 0 | 1 | LEFT SENSOR |
| 0 | 1 | 0 | RIGHT SENSOR |
| 0 | 1 | 1 | BOTH SENSOR |

Output sequence

* Pull down Po (low)
* Write output bits Mo-M3
* Pull up Po (high)

Figure 3.4: Collision detection control specification

Flame Tracking

* IR
* Calibration graphs (Average results over st tests)
* Test every sensor
* Fast output (Below iso ms per output)
* Combined power requirement must be below 9V batteries.

System Output

* Number of pins:5
* Pinout

Po-Read inhibit

P1-Mo

P2-M1

P3-M2

P4-M3 Reserved, leave on low

Output Pattern

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M3** | **M2** | **M1** | **Mo** | **EVENT** |
| 0 | 0 | 0 | 0 | NO HEAT SOURCE |
| 0 | 0 | 0 | 1 | HEAT SOURCE TO THE LEFT |
| 0 | 0 | 1 | 0 | HEAT SOURCE TO THE RIGHT |
| 0 | 0 | 1 | 1 | HEAT SOURCE TO THE REAR (DIRECTLY BEHIND) |
| 0 | 1 | 0 | 0 | HEAT SOURCE TO THE FRONT (NO DISTANCE INFO) |
| 0 | 1 | 0 | 1 | HEAT SOURCE TO THE FRONT (WITHIN RANGE) |

Figure 3.5: Flame detection control specification

To demonstrate that these modules can indeed be integrated, the following finite state machines were also produced at the early stages of the project.

**COLLESSION AVOIDANCE**

**INITIAL**

COLLISION STOP

NO

OBJECT

TURN ON LEFT BOTH RIGHT

TYRN RIGHT TURN TURN LEFT

LEFT

**NO OBJECT**

NO OBJECT

MOTOR STRAIGHT

Figure 3.7: FSM to illustrate collision avoidance.

Throughout the project, these plans were closely followed, and the result is a robot that contained modules without much deviation from the original specifications. This is important, as it means that we can continue with the integration of these modules according to the original FSM, as we will see in section 4.1 and Appendix A.

Note that for software implementation, CCS C compiler and ICD are used to program microcontrollers as CCS compiler is relatively easy to set up and contains debugging features.

### 3.2 Task Assignment

The modules are assigned to individual team members to work on. Task assignments are as follows:

* Motor control & Chassis design: Warwick and Xin

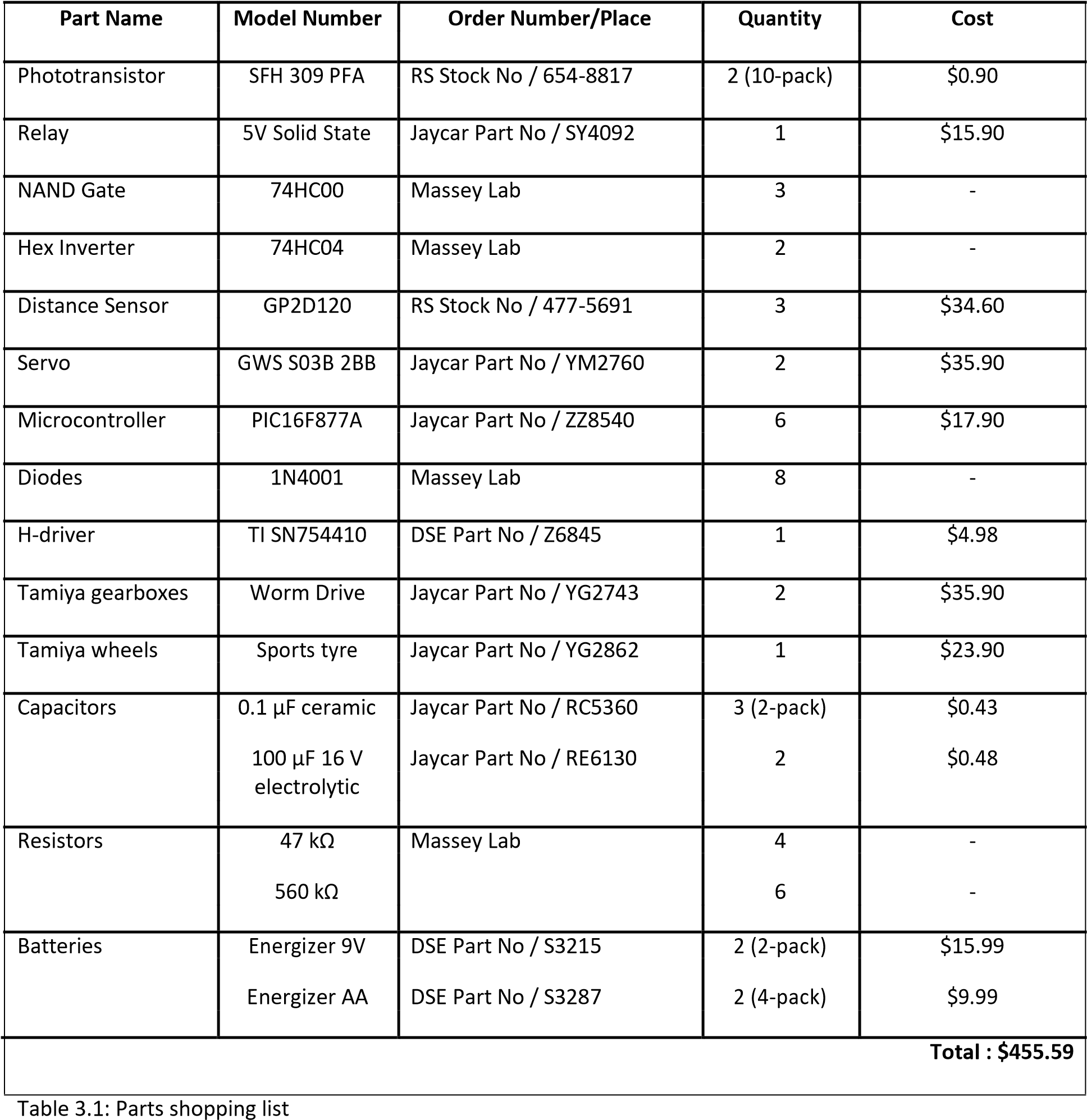
* Proximity control: Nick

* Fire detection: Ayaka

* Master microcontroller, calibration and overall design: James

### 3.3 Parts List

Table 3.1 shows the parts used in our project.



## 4. Systems Design and Integration

### 4.1 Master controller

As the master controller part mainly requires programming and calibration from other parts, the first two sections are ignored.

#### 4.1.1 Hardware

Circuitry wise, the main board is the least complex. Because the main board’s purpose was to take in messages from the sub modules, and output correct fan and motor control messages, all that was required hardware wise was the use of connecters, and parts required to make a PIC function. List one show the parts used for the main board:

* Machined sockets
* 10MHz oscillator
* PIC16F877A
* Programmer Socket

Worth noting here is the use of a 10MHz oscillator, this allows the main chip to operate on a much higher speed fundamentally compared to the sub modules. This means that at a hardware level, the main chip would be able to communicate with all sub modules simultaneously without becoming the bottleneck of the entire system.

Figure 4.1.1 shows the pin connection of the PIC, from this we can see where each of the sub modules are interfaced with the main board.

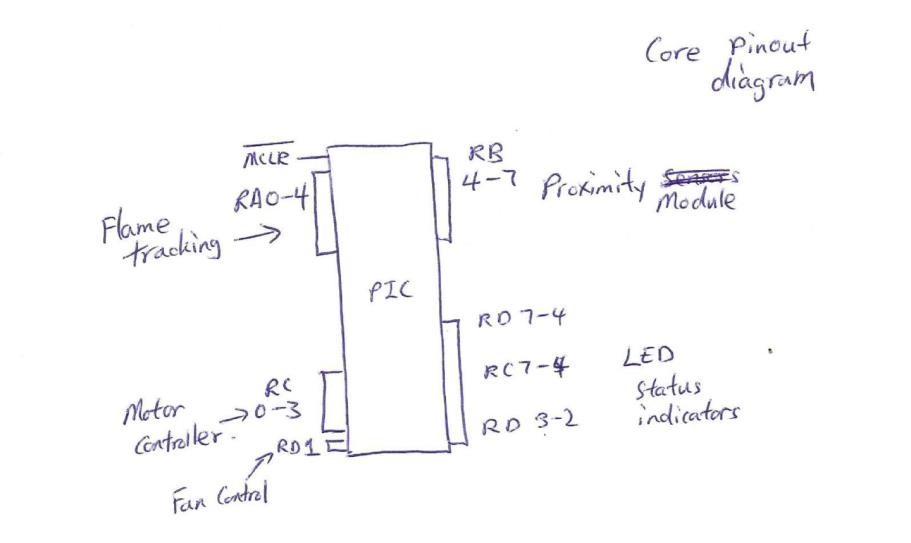


Figure 4.1.1: Core pin diagram

#### 4.1.2 Software

The software of the main chip is the most complex. This is because the main chip has to monitor all sub modules, and perform the correct actions based on external inputs. The actual code is directly derived from the FSMs shown in FIG 3.6 and 3.7, and as such no line by line analysis will be provided here. A copy of the code can be found in APPENDIX A, here we will focus our discussion on some particular problems, and on the RTOS (Real Time Operating System).

##### 4.1.2.1 RTOS

Real time operating system (RTOS) is one in which multiple sub systems (components) are running in parallel, with full data sharing and co-ordination. This is usually achieved by first having a foundation, which can delegate between the different components that needs to run at the same time or use the same resources. The foundation is also responsible for inter-component communication. The figure

4.1.2 shows a typical real time operating system.

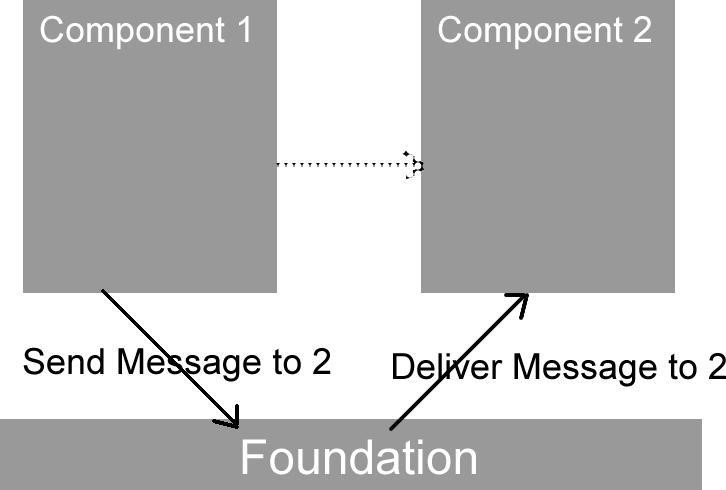


FIG 4.1.2 Typical RTOS Implementation

Here, the foundation can be set to run every 10ms. This is done at the hardware level by setting one of the PIC’s timers to a 10ms interrupt. On top of the foundation there are a number of registered components (2 in this case). One restriction on the RTOS system is that the trigger interval of the registered components must be higher than the foundation’s. Every 10ms, the foundation triggers, it checks to see if any component needs to run at the current tick. If there is, then that component is triggered. If the component wants to send a message to another component, it will tell the foundation about this transaction, and the foundation will take note of it, and deliver it to the destination component the next time that component is triggered.

There are a number of advantages offered by a RTOS. Apart from being inherently free from bottlenecks and being very fast. We can instruct individual RTOS components to turn off as the situation warrants. For example, in our project, we do not need the collision avoidance (CA)

sub module to be running while the robot is rotating on the same spot to face the flame. In this case, we can disable the collision detection RTOS component by using the code:

*Rtos\_disable(rtos\_CA)*

Where rtos\_CA is a registered RTOS component written by us to handle the proximity module inputs.

In our project, the RTOS foundation is provided by CSS as a set of libraries, so we needed to write our mission specific components, and register them.

##### 4.1.2.2 Core software

The core software is separated into the following parts:

* Proximity
* Flame Tracking
* Motor

Each of these parts is implemented as an RTOS component. The motor controller is scheduled to run every 600ms. This large delay is intentional, as by lengthening the interval the motor controller triggers, we can effectively limit the number of transitions the motors can make per second. A very short interval can cause the motors to switch directions very often, and thus destroying the H-Driver hardware.

Of the three components, only the motor driver is enabled at all times. The other two are disabled as required. The basic flow of the system can be summed into the following steps:

For Proximity:

* If there are no obstacles, go forward and turn flame tracking back on

* If there is an object on the left, disable Flame Tracking (and thus entering collision avoidance state), and turn right. Flame tracking is disabled, because we want the robot to clear the

obstacle first, before being guided by flame tracking again.

* If there is an object on the right, again we disable flame tracking (enter CA), and turn left
* If there is an object in front, the default action is to disable flame tracking and turn right

For Flame tracking, the guiding principle is that flame tracking RTOS is only allowed to rotate the robot to face the flame. It is not allowed to actually move the robot forward. This is so that within Flame Tracking, we can periodically disable collision avoidance without the risk of running into an object. The overall logic is as follows:

* If there is no flame source, we want to enable CA, and leave the running of the robot to it

* If there is flame source in front, then we do nothing. CA RTOS should be active, and we want to let that handle the movement of the robot. Note we do not have to worry about CA avoiding the candle, as the range of the flame detector is much further, and the program is designed to stop CA once the candle is within fan range.

* If there is flame source on the left or right, we want to disable collision avoidance, and rotate toward the flame. We need to turn off collision avoidance, as otherwise the CA RTOS may detect the candle as an obstacle, and try to avoid it

* If there is flame source directly behind, then we want to disable CA, and rotate right to face the flame

* If the flame is very close to us and is directly in front, we want to stop the robot, turn on the fan for 10 seconds, then turn around 180 degrees, and enable all the sensors again to start another run

##### 4.1.2.3 Flame Tracking hysteresis

This is one of the very first problems faced by the main program. In fig 3.6**,** we see that we are to turn left or right when a certain set of sensors are detecting a flame source. There are two details that were not expected which appeared when we tested our design.

Firstly, there are gaps in the sensor array, and as such there are small transition regions in which no sensor is seeing the fire. This happens very often when the robot is turning towards the flame. The result of this is that the robot would start turning towards the flame, and would lose tracking when a blank sensor region is encountered and start going forward again. To solve this, we added a hysteresis value into the programming, which directs the robot to continue turning in the direction of the perceived flame even if no sensors are detecting it. This means that if the robot loses the flame while turning towards it, it would have some more time to turn towards the intended direction before the flame source is deemed lost and the robot resumes its normal search pattern again.

Also, **in section x.2.2**, we see that the flame tracking RTOS is turned back on as soon as the robot clears an obstacle. When implemented, this causes the robot to clear an obstacle, and immediately start to turn toward the flame source, only to be redirected by CA again to the opposite direction. To solve this problem, we added a CA hysteresis, which would allow the robot to clear an obstacle, go forward for a little while before turning the flame tracking back on.

##### 4.1.2.4 Right angle turning

Another interesting problem faced by our robot, is that it can get stuck when entering a right angle corner at 45 degrees. This is because as one of the sensors senses the wall and the robot starts to turn, the other sensor also detects the wall, and the robot is now stuck in an infinite loop going left and right. To solve this problem, we added memory to the main program. Whenever a turn is made by CA, the current turning direction is logged, and if the robot is directed to turn in that same direction again having turned from the opposite direction, the robot would instead perform a 90 degree turn to clear the dead corner. For example, if the robot turned left on a right angle corner, it would immediately start to turn right, and the back to turning left. On the last left turn, the program would detect that it is performing a repeated left turn, and would instead turn 90 degrees. The result of this can be seen in videos (see Appendix E).

### 4.2 Motor Control

One of the main components of our robot is the motor controller board. The module is designed to read in commands from the main controller, and apply an appropriate control signal to the chosen motor driver. A PIC16F877A microcontroller by Microchip, combined with a quad half-h driver from TI was used to achieve this.

#### 4.2.1 Decision on parts

We initially decided to use mini DC servos such as those used in hobbyist remote control devices. These servos are composed of a DC motor mechanically linked to an output potentiometer via reduction gears, and are controlled by injecting a PWM (pulse-width modulated) signal wave into

the onboard electronics. The electronics contained within the servo housing determine the desired position from the pulse length of the input waveform, and the motor is powered until the desired position is reached (feedback taken from the potentiometer). In our case the servos were modified to have continuous rotation and apply speed control rather than position control. This was achieved by removing the linkage between the potentiometer and the output gear; effectively fooling the feedback controller into continuously powering the motor until the neutral pulse (1.5 ms in length) was applied. The difference in speed is then directly related to the pulse length of signal wave.

#### 4.2.2 Testing process

Upon testing, however, we decided against the use of servos, as generation of the PWM waveform and speed calibration between the two servos proved tricky to implement in practice. We instead opted to go for a dual H-bridge setup. H-bridges are often used in robotics or other applications that require backward and forward control of DC motors. They can support high speed switching and low driving voltages, and are available in integrated circuit (IC) form.

We tested two H-bridge ICs: one an L293D, as produced by ST Microelectronics; the other an SN754410, produced by Texas Instruments. Both are Quadruple Half-H drivers and provide bidirectional drive for inductive loads such as relays, solenoids, and DC and bipolar stepper motors. The ICs have TTL and CMOS compatible inputs, and offer drive currents of up to 600 mA per channel for the L293D and 1 A for the SN754410. The supported voltage range is from 4.5 to 36 V for both the logic and motor supplies (hardware separated). We chose the latter device for its additional current capacity, as our motors would draw up to 1 A each under stall conditions (this increases with higher than nominal voltages).

Two Tamiya Worm Gearbox H.E. (high efficiency) units were chosen for our drive system. They are supplied with RE-260 motors (3 V), and can be configured for 216:1 and 336:1 gear ratios – we opted for the latter as low speed-high precision was preferred for our application. The gearbox is shown in Figure 4.2.1.

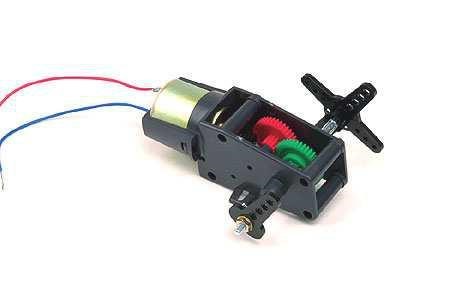


Figure 4.2.1: Tamiya Worm Gearbox with Motor

#### 4.2.3 Hardware

Figure 4.2.2 below displays the chip pin-out diagram for the SN754410. Voltages chosen were 9 V for the logic supply (Vcc1) and 4.5 V for the motor driver (Vcc2). This fed 3.8 V (overdriven) to the motor outputs (1, 2, 3, 4Y) on a fresh set of batteries – approximately 0.7 V is dropped by the internal hardware. Initially we ran Vcc1 on the same 5 V supply as the rest of our logic (microcontrollers), but we ran into problems with the supply voltage dropping below the threshold for this chip and causing erratic operation.

The logic for the SN754410 is also displayed in Figure 2. As can be seen, there are four inputs to the chip, along with two enable pins. The motors will run when the enables are set high, and will be in a high impedance state when set low (braked). We tied the enable pins directed to 5 V for simplicity, as dynamic braking was not necessary for our worm gear configuration.

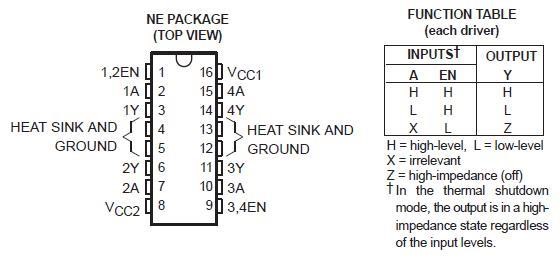


Figure 4.2.2: Pin-out diagram for the SN754410 IC

After integrating all elements of our system together we encountered problems with our PICs resetting upon fast switching of the motors. We discovered that the in-built diodes on the output pins were in fact only designed for ESD (electrostatic discharge) protection, and were not rated for voltage clamping (spike protection) like the ones in the L293D. This issue was addressed by installing diodes of the 1N4001 variety on the motor outputs to correctly clamp the output voltage to within its intended range, and prevent dirty current from leaking into the ground trace (common to all our devices) and causing disturbances. A 470 μF bipolar capacitor was also placed across the motor driver supply, to ensure the voltage remained constant. Figure 3 displays the complete motor controller circuit used.

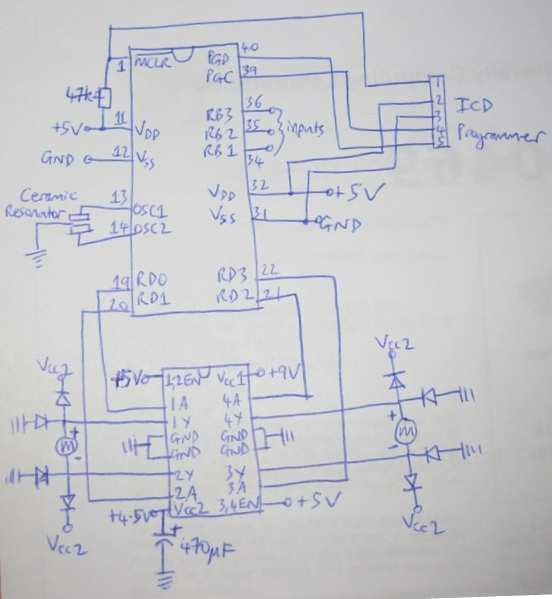


Figure 4.2.3: Complete motor driver circuit

Pins RD0-3 were used as outputs to the H-bridge, and pins RB1-3 were assigned as inputs from the main board. The ICD programmer utilised pins MCLR, PGC, PGD, Vdd and Vss; MCLR was also tied to 5 V through a 47 kΩ resistor, for normal operation. A 4 MHz ceramic resonator was used as the clock input for the PIC.

**4.2.4 Software**

The commands read in by the motor controller were in the following form:

|  |  |  |  |
| --- | --- | --- | --- |
| **RB1** | **RB2** | **RB3** | **Event** |
| **0** | 0 | 0 | Stop |
| **0** | 1 | 0 | Left |
| **1** | 1 | 0 | Right |
| **0** | 0 | 1 | Forwards |
| **1** | 0 | 1 | Backwards |

Functions were created for each event, and involve setting pins RD0-3 either high or low in order to achieve the desired result. All unused pins on the microcontrollers were set to outputs by use of the tri-state registers, so as to avoid floating input pins causing potential problems during operation.

Fuses set were: XT (crystal/resonator <= 4 MHz), NOLVP (no low voltage programming), NOPROTECT (code not read protected), NOBROWNOUT (no brownout reset) and PUT (power-up timer).

The complete piece of code is as shown in Appendix C.

### 4.3 Proximity Control

#### 4.3.1 Decision on parts

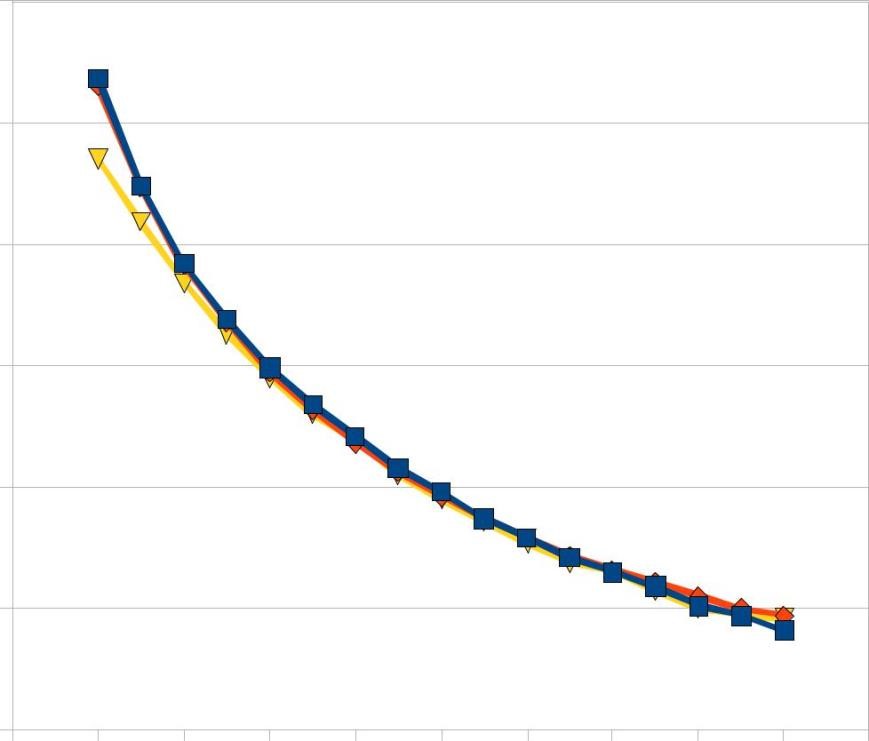
When it came to selecting the proximity sensors, we decided on the sharp proximity sensors as we were familiar with them having previously used the sharp GP2D12 on another robot. We looked at the various sharp sensors looking at the specs, mainly detection distances. In the end we decided on the sharp GP2D120, as it had closer detection than the others 4cm-30cm, if we compare with the detection distances of the GP2D12 that detects 10cm-80cm (“Sharp Rangers”, 2008).

#### 4.3.2 Testing process

Initially we got two sensors and we tested first the distance vs output voltage with different coloured objects (table 4.3.1, Graph 4.3.1).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Distance(cm)**  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **White(volts)** |  | **Red(volts)** | **Candle(volts)** |
| 2.68 | 2.65 |  | 2.35 |
| 2.24 | 2.23 |  | 2.09 |
| 1.92 | 1.91 |  | 1.84 |
| 1.69 | 1.68 |  | 1.63 |
| 1.49 | 1.47 |  | 1.45 |
| 1.34 | 1.32 |  | 1.30 |
| 1.21 | 1.18 |  | 1.19 |
| 1.08 | 1.06 |  | 1.05 |
| 0.98 | 0.96 |  | 0.95 |
| 0.87 | 0.87 |  | 0.86 |
| 0.79 | 0.79 |  | 0.77 |
| 0.71 | 0.72 |  | 0.69 |
| 0.65 | 0.66 |  | 0.65 |
| 0.59 | 0.61 |  | 0.57 |
| 0.51 | 0.55 |  | 0.50 |
| 0.47 | 0.50 |  | 0.47 |
| 0.41 | 0.47 |  | 0.46 |

*Table 4.3.1* Volts per Distance



Voltage (volts)

3

2.5

2

1.5

White(volts)

Red(volts)

Candle(volts)

1

0.5

0

2 4 6 8 10 12 14 16 18 20 22

Distance (cm)

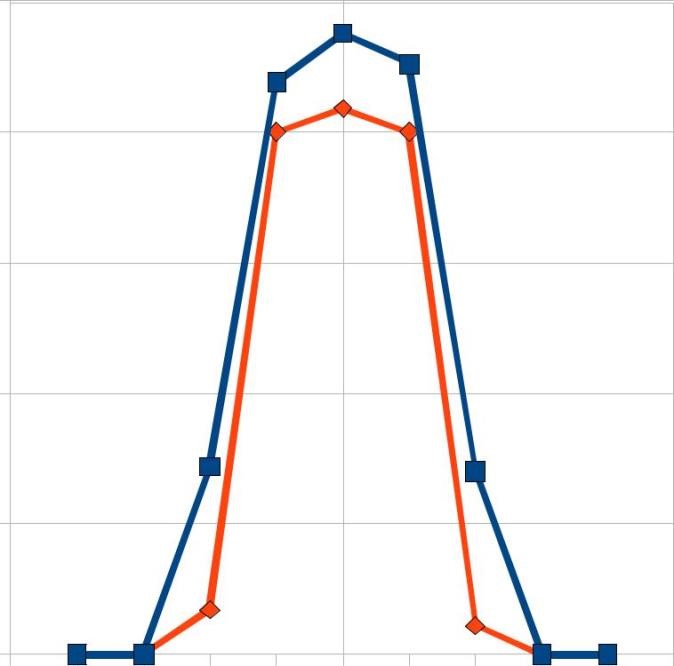
*Graph 4.3.1*

As we can see from the above graph that the values are fairly consistent and the graph is fairly accurate with the graph provided in the datasheet (**Appendix A**), we then tested angle off centre the sensors could detect objects for this I used a candle (Table 4.3.2, Graph 4.3.2).

|  |  |  |
| --- | --- | --- |
| Angle from Centre (degrees) | Voltages at 4cm | Voltages at 5cm |
| -40 | 0 | 0 |
| -30 | 0 | 0 |
| -20 | 0.72 | 0.17 |
| -10 | 2.19 | 2.00 |
| 0 | 2.38 | 2.09 |
| 10 | 2.26 | 2.00 |
| 20 | 0.70 | 0.11 |
| 30 | 0 | 0 |
| 40 | 0 | 0 |

*Table 4.3. 2*

Volts per Angle



Voltages (volts)

2.5

2

1.5

1

0.5

0

Voltages at 4cm

Voltages at 5cm



-50 -40 -30 -20 -10 0 10 20 30 40 50

Angle (degrees)

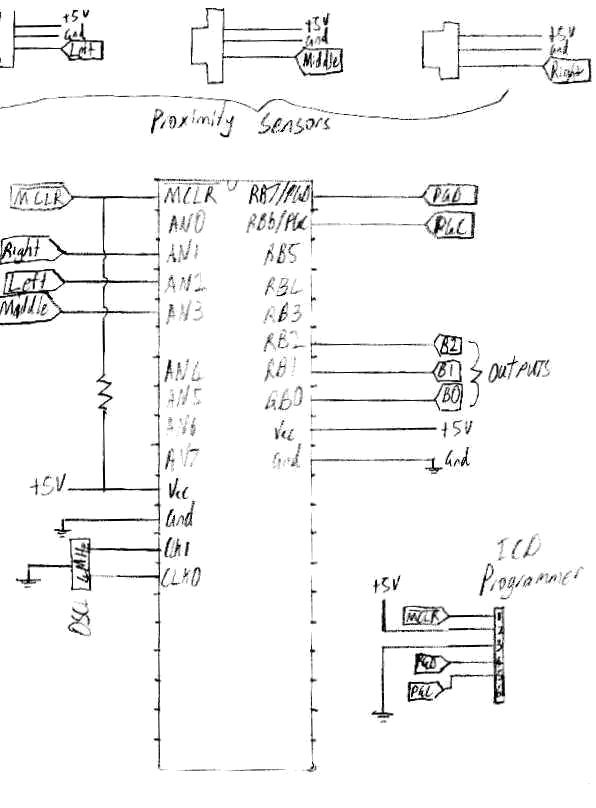
*Graph 4.3.2*

From the above data we see that the angle off centre the sensors can detect at is around 100, because of this we decided on acquiring a third sensor so the robot doesn’t drive into a thin object, i.e. a candle.

**4.3.3 Hardware**

#### Circuit Design Process

When we designed the proximity control circuit, we also designed it to include a programmer so we could program the PIC microcontroller, to find out how to solder up a programmer we looked at a circuit diagram of a development board that includes the PIC we used (the pic16F877A) and crossed referenced with the pin diagram of our PIC (found in the datasheet for that PIC). Because the output of the sensors was analogue we had to convert them to digital, therefore we used the A/D inputs in the PIC for the proximity sensors (AN0 – AN7).



**Figure 4.3.1: Proximity Control Circuit**

When we tested the above circuit it kept resetting to overcome this we added a bypass capacitor on each of the sensors output to ground, this stabilised the circuit.

#### 4.3.4 Software

The purpose of the proximity control was to output the following code:

|  |  |  |  |
| --- | --- | --- | --- |
| **RB0** | **RB1** | **RB2** | **Event** |
| **0** | 0 | 0 | No Object |
| **0** | 0 | 1 | Left or Mid and Left |
| **0** | 1 | 0 | Right or Mid and Right |
| **0** | 1 | 1 | Mid or Mid and Left and Right |

The reason for the three outputs is that the third output RB0 is used to tell the receiving PIC when data is being sent and when the data has finished being sent for each event, i.e. RB0 is set to low whilst data is being sent, once finished RB0 is set to high.

Firstly I wrote a drive for the three sensors that initialises the ADC’s and gets the data from the three sensors and converts and stores the digital outputs, this code is shown in Appendix D.

### 4.4 Fire detection control

The purpose of the fire detection panels is to find the fire (candle light) and tell the master processor the direction of the fire. The front panel also need to find if the fire is close enough to put the fan on or not.

#### Initial concept overview

The initial idea of finding the direction of the fire is to use hexagon panels as explained in the project specification (Section 3.1). Six panels are prepared with one or more fire detecting sensors and calibrated together in a hexagon shape. In this way, the body of the robot does not need to rotate around to find the fire. In fact, the fire can be found in any direction no matter which direction the body of the robot is facing.

For the sensor panels to detect the direction of the fire fast and easily, the original idea was to read in each panels as binary by processing values through some basic logic gates. As you can see in the next sections however, it is not possible to use logic gates as static threshold value must be chosen.

The Analogue to Digital Converter (ADC) on the microcontroller (PIC 16F877A) is used to correctly find the direction of a flame. The ADC can read the voltage output from each sensor accurately but takes some time. For the PIC16F877A microcontroller, we must wait few microseconds before it can start reading the value from the sensor. Then it takes a minimum of 16 microseconds per sensor. Therefore, to read in all six sensor values to the microcontroller means it takes at least 100 microseconds.

#### 4.4.1 Decision on parts

We require fire detecting sensors which provide us with accurate readings of the flame direction and relative distance from the robot. Optionally a non-contact temperature sensor may become handy to determine the flame’s distance from the robot. There are different types of sensors we can use for fire detecting panels. There are:

* Phototransistors: Light is detected through a transistor’s base. If light is detected, the output

voltage to be high and vice versa. A wide range of products is available from RS-Online NZ.

* Pyro-electric sensors: Similar to IR sensors but with many useful processing components integrated. Usually used in the motion detector but can be used as flame detector as it uses multiple IR sensors to detect temperature changes. Some available from RS-Online NZ.

* Hamamatsu UVTron flame sensor: Sensitive to flames as it measures photons in the UV spectrum. Acroname Robotics has the UVTron as well as an UVTron drive circuit.

From the list, we decided to use phototransistors as a wide range of them were locally available as well as internationally, whereas other two were only available internationally. In addition, the cost of phototransistors in general was much cheaper than the other two. This is a big advantage because each panel can have one or two sensors to increase the accuracy of finding the direction of the flame. Also, we would have plenty of spares for the emergency cases.

From the wide variety of phototransistors, the phototransistors with a large wavelength range and large range of acceptance angle are examined. We want the phototransistors with a large

wavelength range because the candle flame has a wavelength of around 600 to 1000 nano meters. Also, the wide range of acceptance angle allows us to cover all 360 degrees view from the robot.

The final choice was made to use SFH 309 PFA manufactured by OSRAM Opto Semiconductors. This phototransistor has the small wavelength range at 880 nano meter and has the acceptance angle of 150 degrees. It also has the rise/fall time of 6 microseconds, which was the best out of our selections.

**4.4.2 Testing process**

The sensor is tested with a candle light to examine its capabilities using a simple circuit shown in figure 4.4.1. Its output is examined for different distance and directions to the flame.

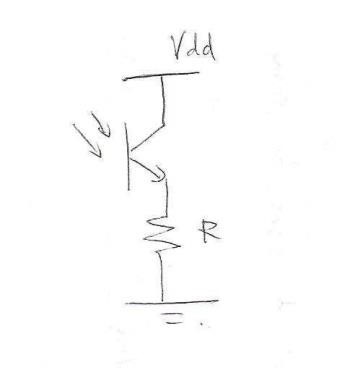


Figure 4.4.1 : Circuit for the phototransistor testing.

When a sensor was tested for the first time, the sensor reacted to a candle light but the output value was very small. In order to attain better output voltages, different resistors needed to be experimented as well as the distance and direction.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Output voltage (V) |  | |  |  |  |  | Distance (cm) | | |  |  |  |  | |
| 10 | | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 |
| Resistor  (Ω) | 470 | 0.37 | 0.37 | 0.35 | 0.33 | 0.31 | 0.28 | 0.19 | 0.12 | 0.07 | 0.04 | 0.03 | 0.01 | 0.01 |
| 220k | 4.78 | 4.76 | 4.74 | 4.68 | 4.63 | 4.37 | 3.97 | 3.36 | 3.01 | 2.48 | 2.12 | 1.66 | 1.48 |
| 560k | 4.88 | 4.88 | 4.85 | 4.83 | 4.82 | 4.70 | 4.44 | 3.60 | 3.00 | 2.90 | 2.43 | 2.10 | 1.90 |

Table 4.4.1: Experiment result of the above circuit. The result was recorded in a room environment.

The results recorded on this table were collected in an engineering lab one afternoon. The blind was shut to avoid sunlight so the result can be assumed under a normal room environment with fluorescent lights. This table clearly shows that as the distance increases between a candle flame and a sensor, the output voltage decreases. With different resistance values the distance differences are more accurately measured.

Next, the outputs under different light environment were tested using the resistor of 560 kilo ohms. The table 3 illustrates the output.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Output voltage (V) | | Distance (cm) | | | | | | | | | | | | |
| 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 |
| Lighting | Dark | 4.88 | 4.88 | 4.85 | 4.83 | 4.82 | 4.70 | 4.44 | 3.60 | 3.00 | 2.90 | 2.43 | 2.10 | 1.90 |
| Light | 4.88 | 4.87 | 4.86 | 4.84 | 4.82 | 4.80 | 4.54 | 4.38 | 4.07 | 3.99 | 3.86 | 3.90 | 3.88 |

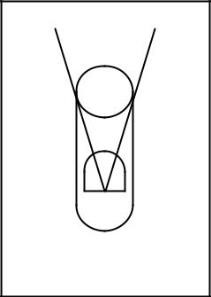
Table 4.4.2: Output voltage for different light environment using a resistor of 560 kΩ.

The output voltages in two different environments are similar when the distance to the candle is close. As the candle flame goes further from the sensor, the output voltage drops more dramatically in the dark lighting environment. Whereas in lighter environment, the output voltage stays quite high despite of the distance from the flame. This is expected as the sensor goes further from the flame the other light sources such as sunlight interferes with the sensor.

Through this experiment, it was proved that the phototransistor could detect a candle flame well from some distances. However, its reading varied with the light environment therefore it is necessary to build some form of shield for each sensor. It was also rather empirically found that the resistance of 560k ohm would give the desired output voltages from sensors.

**4.4.2.1 Cover design.**

To limit each sensor’s vision from the sunlight and room lights as much as we can, it is necessary to create a cover similar to the shield on traffic lights for each sensor.



First of all, to increase the sensor’s focus in one point, a dummy cap is created for the sensor (see Fig 4.4.2). The dummy cap was a black cylinder which fitted on the sensor as shown here. This increased its focus and so detected the direction of the flame more accurately. However, it was at the same time too limiting considering that the sensor can cover 150 degrees. Also, it should be taken into account when all six sensors are put together; each sensor must cover at least 60 degrees of the view to browse 360-degree view from the robot

Figure 4.4.2:First cover The second prototype of the cover is shown in figure 4.4.3. The first diagram (a) design. shows the vertical view of the cover. Considering the sensor is placed at the centre, this cover limits the sensor to 15 degrees of top view and 30 degrees of the bottom view. In this way, lights from above such as the sunlight and bright-coloured objects can be eliminated. The second diagram (b) shows the horizontal view of the cover. As mentioned earlier, this cover should give the total of 60 degrees view to the sensor, allowing when all six sensors are put together it covers 360 degrees.



5.9

cm

3.7

cm

40

˚

3

cm

˚

60

3

cm

a

b

Figure 4.4.3: Cover design for a sensor. a) Cover viewed from side. b) Cover viewed from top. Note that the above is not in scale.

The second prototype was tested on the same circuit using the resistor value of 560 kilo ohms. The result of the output voltage is again recorded on the table below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Output voltage (V) | | Distance (cm) | | | | | | | | | | | | | |
| 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 |
| Lighting | Dark | 4.88 | 4.88 | 4.85 | 4.83 | 4.82 | 4.70 | 4.44 | 3.60 | 3.00 | 2.90 | 2.43 | 2.10 | 1.90 |
| Light | 4.88 | 4.87 | 4.86 | 4.84 | 4.82 | 4.80 | 4.70 | 4.10 | 3.40 | 2.90 | 2.60 | 2.30 | 2.14 |

Table 4.4.3: Output voltage for different light environment using the prototype cover.

The table 4.4.3 shows the resultant output voltage when the sensor with the cover was tested in different lighting environment. As we observed earlier in table 4.4.1, the output voltage was very similar when the sensor was placed near to the flame. As the separation increased, the output should decrease but in lighter environment the output voltage decreased to certain amount and reached a stable voltage due to the lights. With the prototype cover, the output voltage successfully decreased as the separation from the flame increased even in the light environment.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Output voltage (V) | |  | Angle (degrees) | | | | |
| 0 | 10 | 20 | 30 | 40 | 50 |
| Distance (cm) | 40 | 4.84 | 4.83 | 4.82 | 4.82 | 2.1 | 1.9 |
|  | 50 | 4.82 | 4.79 | 4.45 | 4.72 | 1.8 | 1.8 |
|  | 60 | 4.82 | 4.8 | 4.1 | 3.6 | 1.7 | 1.7 |

Table 4.4.4: Resultant output voltage of the above circuit for different angles.

Table 4.4.4 shows the output voltage at three different distances. The sensor was turned away from the candle by certain degrees and the output voltage of the circuit in fig 4.4.1 is measured. When the sensor was separated from the flame for 40 cm and 50 cm, the flame was correctly detected within the 30 degrees angles. When the separation between the flame and the sensor is 60cm, the output voltage is wary as it gives us 3.6 volts when the sensor is 30 degrees turned away from the flame. However the voltage difference when the sensor is turned 30 degrees and 40 degrees is big enough to say that the prototype cover successfully limits the sensor’s vision to 30 degrees one way; total of 60 degrees. This proves that the prototype cover is designed with accurate vision.

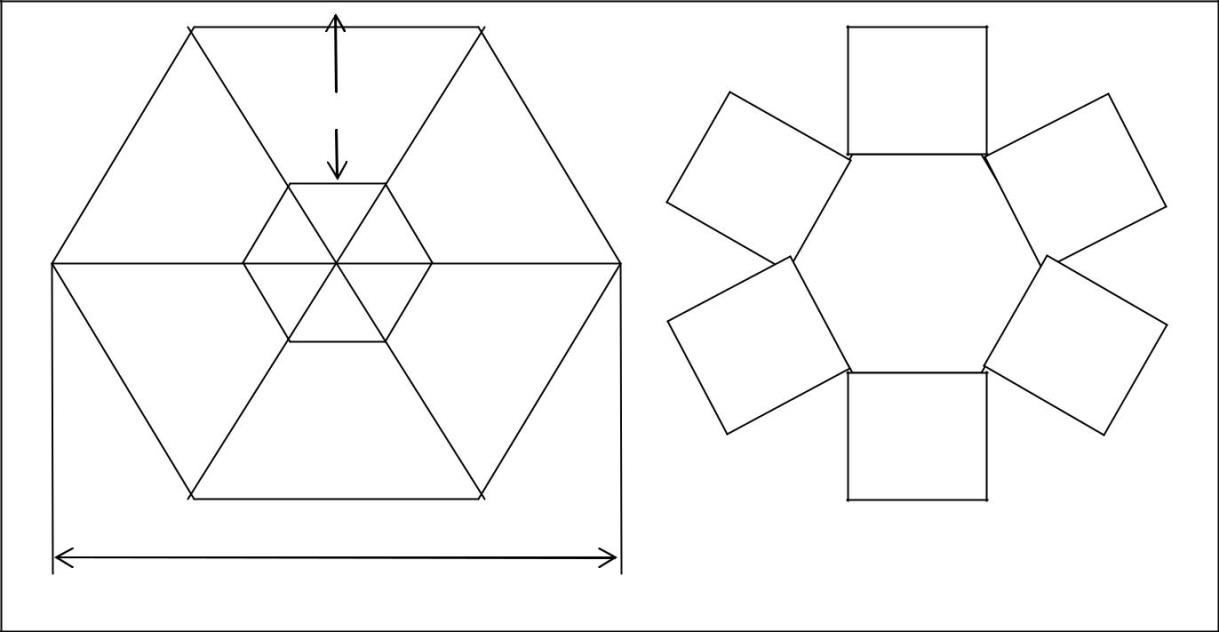
These above two tables show that the prototype cover successfully limited the sensor view. The risk of it is that when the candle light is placed above the sensors, that light cannot be detected. Also, as you can see from the table, the range of output voltages in the dark environment is much lower than the range in the lighter environment. Therefore it may be difficult to implement the fire detecting panel with comparators and logic gates as I described in the initial concept, as the threshold voltage for comparators need to be different for different lighting environment.

#### 4.4.3 Hardware

4.4.3.1 Sensor terminal

As described in the previous section, a successful sensor cover was designed. This section describes the design for all six sensors using the previous cover.

Figure 4.4.4 shows the measurements for the top and the bottom parts of the sensor cover. These measurements are based on the cover shown in Figure 4.4.5.



4.2

cm

2.6

cm

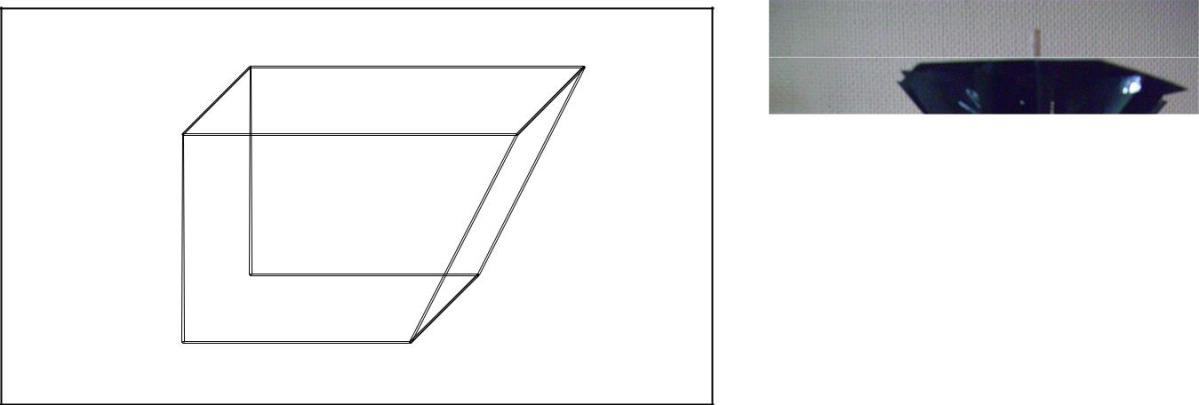
2.6

cm

13.7

cm

Figure 4.4.4: Top and bottom design of six sensor cover



2

cm

4.2

cm

2

cm

2.6

cm

Figure 4.4.5: Cover measurement for individual sensor.



Figure 4.4.6: Photo of the first 26

sensor terminal.

Initially to test this whole cover for all six sensors, sensors were combined to the cover as shown in the photo below. All wires were then connected to the circuit board where resistors were all connected parallel as seen in figure 4.4.6.

Using this model, the output voltage from each sensor was measured for different distance and angles.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| V out | |  | |  |  |  | Distance | |  |  |  |  |
| 10 | 4.87 |  |  |  | 15    20    25    30    35    40    45    50    55    60                              4.84    4.81      4.68    4.59    4.50    3.87    3.35    3.00    2.54 |  |  |  |  | 65 |
| sensor | 0 | 4.87 | 2.10 |
| 1 | 4.85 | 4.84 | 4.82 | 4.78 | 4.72 | 4.63 | 4.57 | 3.99 | 3.49 | 2.88 | 2.47 2.68 |
| 2 | 4.87 | 4.86 | 4.84 | 4.80 | 4.77 | 4.67 | 4.60 | 4.12 | 3.75 | 3.25 | 2.82 2.04 |
| 3 | 4.88 | 4.87 | 4.86 | 4.82 | 4.75 | 4.69 | 4.61 | 3.80 | 3.03 | 2.46 | 1.97 1.58 |
| 4 | 4.86 | 4.86 | 4.83 | 4.77 | 4.64 | 4.53 | 4.44 | 4.24 | 3.85 | 3.64 | 3.34 2.97 |
| 5 | 4.88 | 4.88 | 4.86 | 4.79 | 4.71 | 4.62 | 4.52 | 4.04 | 2.89 | 2.75 | 2.48 1.86 |

Table 4.4.5: Result table for each sensor



Table 4.4.6: Result output voltage for different angles

From these experiments, it was found that the “vision” of each sensor was not limited as well as the previous test. This could be due to the direction of each sensor and error on the cover. As you can see from the photo above, each sensor was placed approximately at the centre but its direction was not directly towards the front. Also as sensors were directly  soldered to wires, it was possible that one sensor was touching another shortening one output value.

Six proper panels were therefore necessary to test the functionality of the cover properly. Also the cover needed to be made out of stronger material. Therefore the same design was constructed out of corrugated cardboard. The cover was made out of black cardboard to avoid as much reflection as we can. Then six panels were made on 2cm by 2cm circuit boards. The panel’s circuit diagram is shown below.

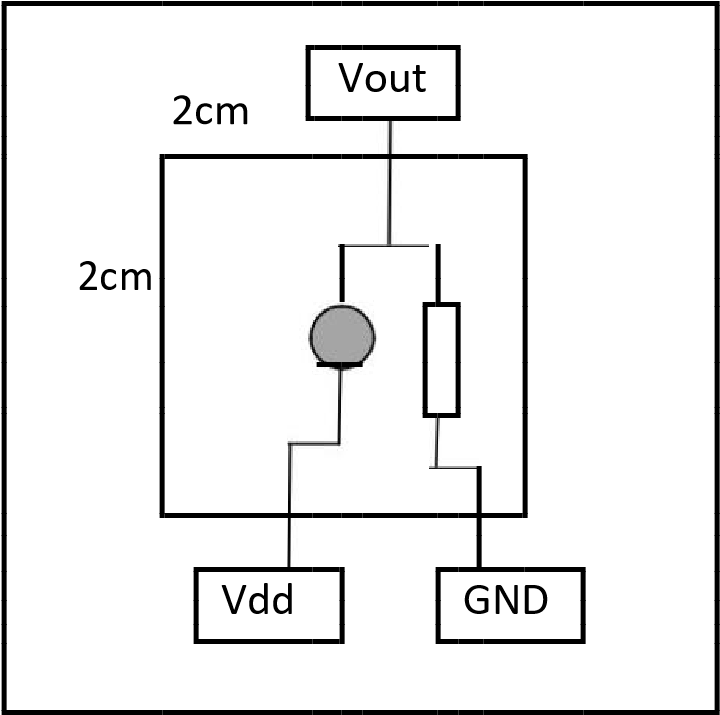
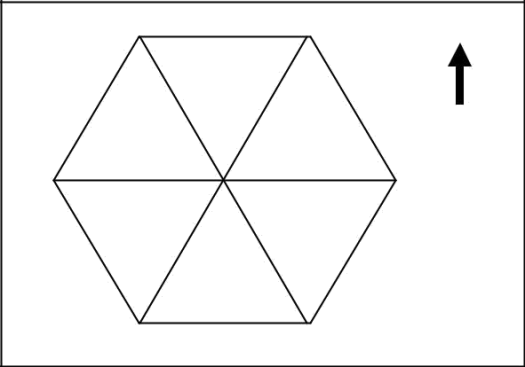


Figure 4.4.7: Circuit diagram for each panel.



0

1

3

2

4

5

Each panel are also numbered from 0 to 5 according to its direction relative to the body of the robot. As shown in figure 4.4.8, 0 indicates the front of the robot, 1 and 2 indicates the left hand side; 3 and 4 indicate the right hand side and finally 5 indicates the directly behind.

Figure 4.4.8: Numbering on each sensor relative to the direction of the robot.

Six panels are soldered and put together to form a sensor terminal. The sensor terminal is held up at some height to allow good vision of the surroundings. When the output voltages from panels were measured again in a similar manner following results were discovered.

1. Output voltages were measured slightly high because of the lights sneaking through the breadboards’ holes. These were blocked by sticking black tapes from behind.
2. There were small gaps between each covers as well. These were again blocked by the use of glue gun and black tapes.

Table 4.4.7 shows another experiment result when the flame was placed 40cm from the front of the robot (i.e. The 0th sensor should be detecting high voltage) in two different lighting environment.

|  |  |  |  |
| --- | --- | --- | --- |
| Light | | Dark | |
| Sensor | V out | Sensor | V out |
| 0 | 4.74 | 0 | 4.52 |
| 1 | 2.67 | 1 | 2.34 |
| 2 | 1.87 | 2 | 0.68 |
| 3 | 1.96 | 3 | 0.23 |
| 4 | 1.34 | 4 | 1.25 |
| 5 | 1.57 | 5 | 0.76 |

The table shows the sensor 0 detecting the flame as it is higher than 4v. When the average voltage value is calculated excluding the highest value, it is obvious that the light environment has higher average voltage value. Using the average values we can compute threshold voltage to see which sensors are detecting the flame.

The differences between the highest value and the average

|  |  |
| --- | --- |
| Table 4.4.7: Result voltage when the light source is directed at sensor 0 separated at 40cm. Average voltages are 1.88V and 1.05V for light and dark environment respectively. | in this record are 2.86 and 3.47. However a numerous records were taken aside from what is shown here and it was found that it is best to estimate the threshold voltage to be 0.8 V higher than the average voltage. Also to estimate the lighting environment more accurately, the second to highest voltages should be ignored just in case the robot can “see” two flames. |

As a result of careful design and experiment, the ideal sensor terminal was constructed that could successfully detect the direction and relative distance to the flame. In addition, by examining the sensor values, the best estimated threshold values are found.

**4.4.3.2 Circuit design for PIC 16F877A**

The output voltages from six sensors were computed using PIC16F877A to tell the master microcontroller of the direction of the flame. As explained earlier, it is not possible to use comparators and logic gates to use the sensor outputs as binary because of the different outcome in different lighting environment. Therefore, six sensor values need to be connected to the Analogue to Digital Converter enabled pins. That is, port A pin 0, 1, 2, 3 and 5 and port E pin0. Master microcontroller is expecting four input pins from the fire detection panel. I chose Port B pin 0, 1, 2 and 3 for the output.

The diagram shows the design of PIC 16F877A. Note that 4MHz crystal was used and its Vdd value is

5 volts.

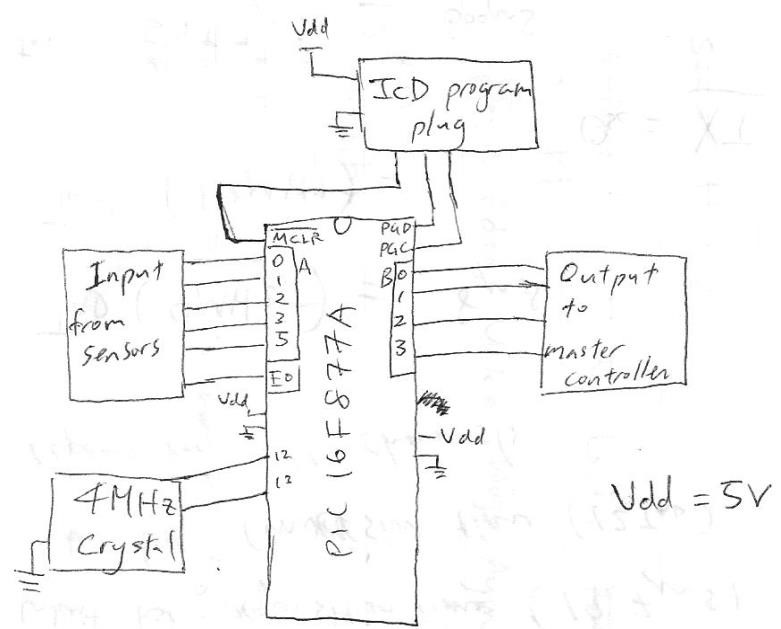


Figure 4.4.10: Pin diagram for PIC 16F877A of fire detection

Also the table 4.4.8 and 4.4.9 summarises the input and output of the above microcontroller.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sensor number | 0 | 1 | 2 | 3 | 4 | 5 |
| Port/pin | A0 | A1 | A2 | A3 | A5 | E0 |

Table 4.4.8: Input to the microcontroller.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | B0 | B1 | B2 | B3 |
| No source | 0 | 0 | 0 | 0 |
| Left | 0 | 0 | 0 | 1 |
| Right | 0 | 0 | 1 | 0 |
| Behind | 0 | 0 | 1 | 1 |
| Front | 0 | 1 | 0 | 0 |
| Fan | 0 | 1 | 0 | 1 |

Table 4.4.9 : Output to the Master microcontroller

#### 4.4.4 Software

**4.4.4.1 Pseudo code**

Pseudo code for the fire detection panel is as follows.

1. Initialise and set up.
2. Read in the voltages from six sensors through ADC
3. Sort the numbers using bubble sort. This makes the later steps easier.
4. Calculate the average measured voltage using the first three sorted numbers (three smallest values). This average value would tell us the approximate lighting environment the robot is in

as seen in the hardware implementation section.

1. Based on the average value, threshold voltage is calculated (VTh = VEnv + 0.8V).

1. Compare the threshold voltage and sorted sensor voltages in sequence. The first sensor that has larger voltage than the threshold voltage is taken for the output. This way, even if the robot sees two candle flames, it would only deal with one flame at a time. Also, for the robot to assume the light accurately, the sensor value must be at least 3 volts in order to say that

that sensor is detecting a light.

1. Output to the master microcontroller accordingly. Refer to the table 4.4.9 for each output.
2. Repeat from step 2.

The above procedure is implemented using C. Refer to appendix B for the actual code.

4.4.4.2 Testing of software

The code was debugged and tested using multi meters and LED lights for outputs.

Firstly, the ADC values and actual output voltages from sensors are compared. It was discovered that there were leaking voltages during ADC process. This was due to poor soldering on panels. When this was fixed, the ADC value for each sensor was again tested and compared. These values never matched, varying by approximately 0.2 voltage difference from the actual value.

When the above problem was solved, the code was debugged with an actual flame. The result showed clearly that the flame was detected by correct sensors and outputted the correct signals at Port B.

### 4.5 Fan controller design

After some discussion, we decided to use a simple on/off fan for putting out the flame. This is because

* It is easy to implement. The rest of the project is already very complex, and as such we want

the fire fighting part to be simple and robust

* Proven to work. A number of previous robots featured fans, and they all seemed to be adequate in putting out a candle flame

The basic fan design has only 3 components:

* 5V low current switching DC relay

* 9V battery that is separate from the rest of the robot
* High speed motor with a fan blade

The reason for using a relay is because they are robust devices, and are tolerant to a wide range of operating conditions. The motor will be drawing a maximum of 1A current, which is difficult to provide with transistors and PICs. Also, we chose a relay with low switching current, as our PIC is only capable of driving up to 20mA on output pins.

For power, we are using a separate 9V battery. This is because our fan motor draws a large amount of current, and our robot’s utility board (section 4.6) is already supplying a number of power devices. The 9V used in the utility board also drives all logic circuits through a regulator, and we want to keep that as steady as possible.

The final circuitry of the fan controller is shown in fig 4.5.1; we see that as the core controller outputs a logic high the relay switches, turning on the fan.

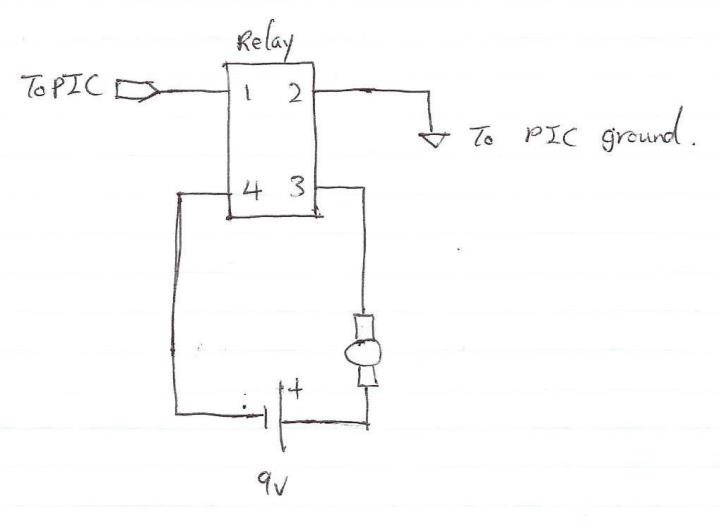


Figure 4.5.1: Circuit diagram for fan control

### 4.6 Utility Board

The utility board is done using the PCB machine here at Massey. The board provides all the power rails for all modules of the robot. As figure 4.6.1 shows, the board contains the following items:

* 9V battery
* 5V Voltage regulator (9V Input, 1.2A Maximum)
* 4.5V battery
* 9V power rail
* 5V power rail
* 4.5V power rail
* STDP (Single Throw Double Pole) Switch
* Ground rail
* Bypass capacitors (0.1uF)
* Spike capacitors (100uF)

#### 4.6.1 PCB Manufacturing

As mentioned before. We manufactured the PCB here at Massey using the PCB printer. The design of the board is done in Altium, and the result is exported to gerber file, which can then be used by the machine to print the board.

Using Altium, we can define a number of layers for the board. In our case we used only the top side, and the bottom side. The top side is mainly used for labelling purposes, where as our bottom side contained all the solder mounts and the traces. Figure 4.6.2 shows the different images of our board design.

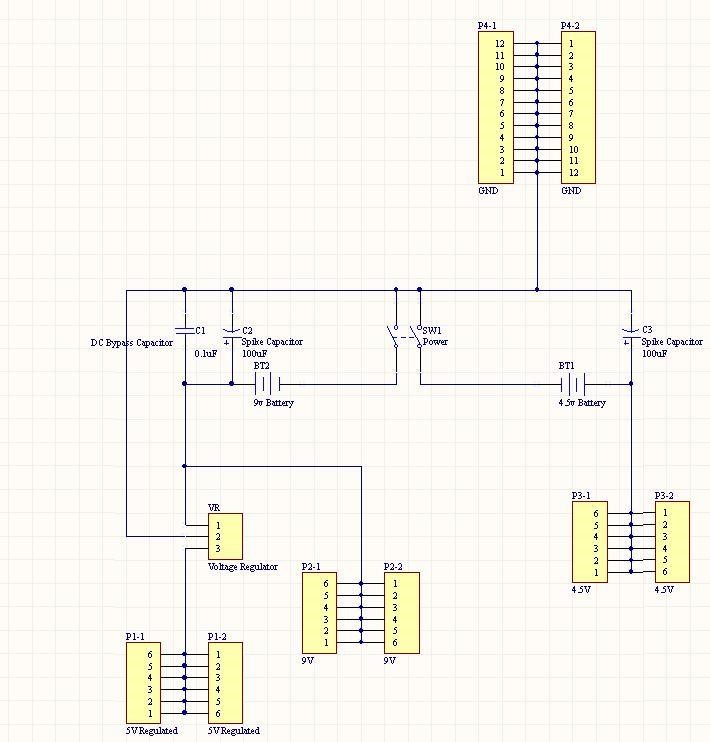


Figure 4.6.1: Circuit design of the utility board

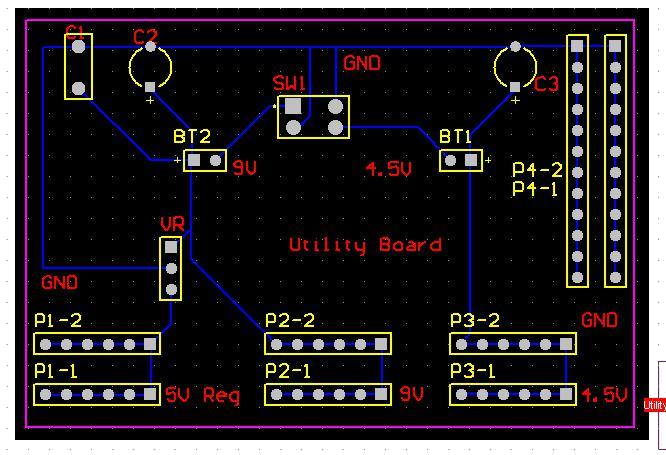


Figure 4.6.2: Different view of the utility board

To design the board in Altium, there are a few simple steps:

1. Start by defining the board dimension

1. Drag and drop parts from the Altium library. For example to add the power rails, simply drag and drop the m slot Machined slots onto the board. Solder mount and drill holes are

automatically calculated based on the part placed

1. Place all the components in the positions desired
2. Draw traces on the bottom layer. The trace is what connects one component to the other
3. If required, place text labels on the top layer, indicating the different components and rails
4. Export the project, and feed the result file to the PCB printer

### 4.7 Chassis Design

Our chassis was designed to be a single layer circular sheet of aluminium with room to accommodate all of the necessary hardware on board. The elements include:

* Four circuit boards plus power board
* Batteries (2 x 9V + 3 x AA)
* Gearboxes/Wheels
* Proximity Sensors
* Fan + Motor
* Tower containing the six phototransistors
* Other accessories including relay and LED board for debugging

The circuit boards were slotted vertically into ‘Tractor Grip’ PCB board supports to hold them firmly in place, while offering ease of removal.

L-brackets for the three Sharp proximity sensors were fashioned from aluminium sheet steel, with the sensors mounted vertically and parallel to one another.

A section of material was removed from either side of the chassis so that the wheels attached to the two Tamiya gearboxes were set inside the outer circumference. These gearboxes were then bolted to the chassis through pre-drilled holes. Rubber bands were applied to the Tamiya sports tyres attached, as we found the semi-slick treads – while looking the part – weren’t providing enough grip on the slippery laboratory surfaces.

Front and rear foam guide wheels were added to keep the robot balanced, while allowing slippage across the floor surface during turns. These also had greater aesthetic qualities than the ping-pong balls some of the other groups were using.

The hexagonally shaped flame tower was glued into place, as it seemed the most effective way to adhere cardboard. The relay and LED boards were then attached to this with double-sided tape, which provided a semi-permanent fix.

A bracket was fabricated for the motor/fan unit to position it at a suitable height for extinguishing candles (approximately 14.5 cm from the ground).

Finally, the batteries were taped in place, along with the main power switch.

CAD Models of the build-up are shown in Figures 4.7.1-4.7.4:

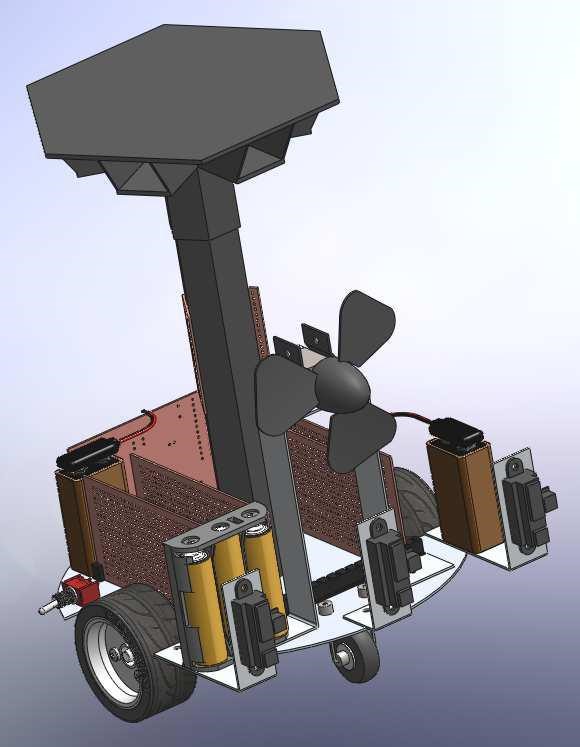


Figure 4.7.1: Front view of robot

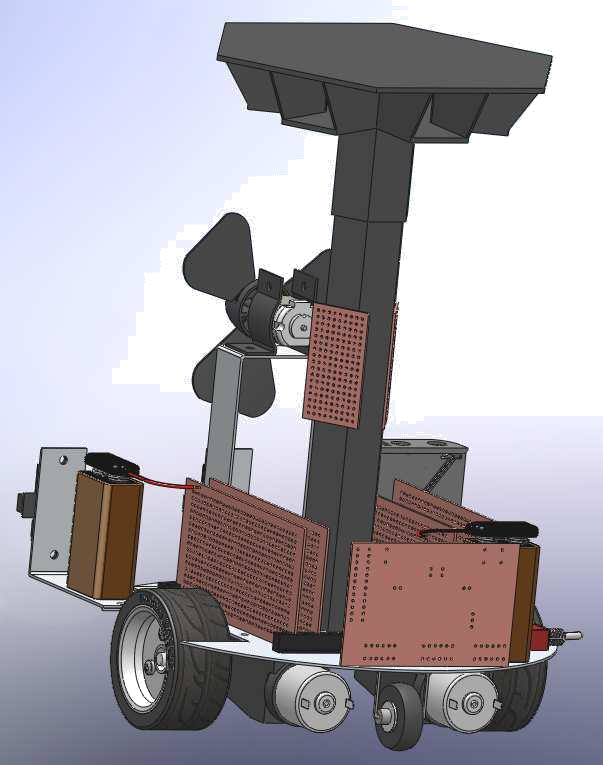


Figure 4.7.2: Rear view of robot

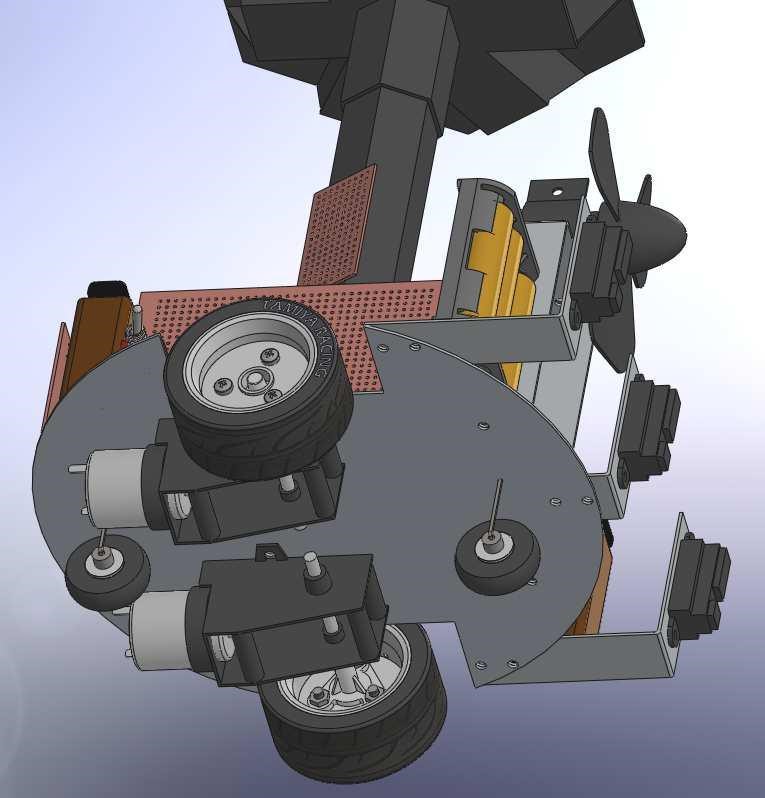


Figure 4.7.4: Robot undercarriage

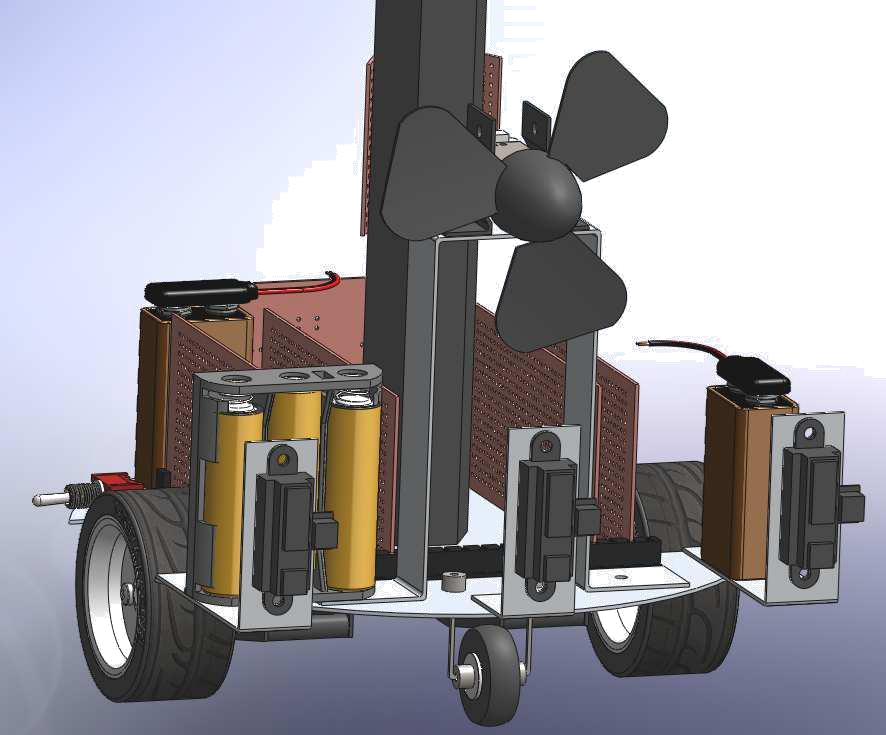


Figure 4.7.4: Detailed view of front section

### 4.8 Integration steps

Here describes the order of modules integrated.

1. Motor control built on the chassis is connected with the master control.
2. Proximity control and 1.
3. Utility board and 2
4. fire detection control and all of above
5. Fan and the above

Note that sufficient testing was done every time a module was added. As you can see from the above order, the base of the robot was integrated first to ensure the robot’s mobility. Proximity control is next added to guarantee the robot’s manoeuvrability around objects. Utility board was the next module to be integrated as the power source for three boards necessary on the robot. Finally the flame detection module and fire fighting (fan) are added.

We faced many problems during integration process. Some problems were:

* The main problem we had was dealing with unstable voltage values at different circuit boards. To stabilise it as much as we can, we added capacitors to ensure constant voltage.

* As mentioned in section 4.2.3, PICs were resetting upon fast switching of motors. To discover the reason took some time but we eventually found that the built in diodes were not for spike protection.

To minimise confusion and inefficiencies and to make later modifications easier, the following points are maintained during the integration process.

* We used machined pins for all connections, the advantage is plug and play, we can plug in different modules with all sorts of devices such as power supplies, signal generators, oscilloscope etc

* All wires are coloured to specific meanings, red = 5v, yellow are signal, ribbon cables are signal, green 4.5v blue 9v
* Robot has a set of slot rails for easy plug/remove of boards
* All boards are cut to specific size to fit in our robot
* The motor would mysteriously reset all of our chips, solved with clamping diodes

The final robot is shown in figure 4.8.1.

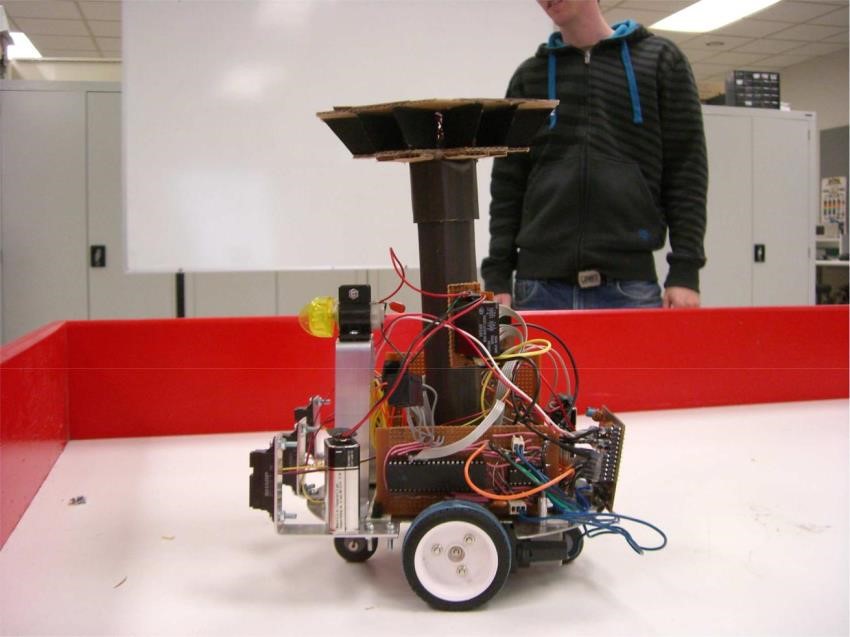


Figure 4.8.1: Final robot

More photos and short video clips can be viewed in the folder called **“folder name”** in our CD as stated in Appendix E.

## 5. Evaluation

We see that some design principle paid off for the project:

* By not allowing the robot to stop, we have produced a very responsive unit (Respond to any external event within 500ms). The Robot does not stop under any circumstance, and can deal with difficult situations such as entering a right angle corner at 45 degrees

* By using cheap components, we were able to incorporate 6 individual panels for fire detection, costing under $40, compared to a single specialized fire detector costing up to

$100. We were also able to absorb 3 broken PICs without going over our budget.

* The use of capacitors is vital to parts that are sensitive to voltage drops. As we found out with all of our PIC boards, adding a capacitor between power and ground can prevent the PIC from randomly resetting due to sudden voltage drops caused by components somewhere else.

38

## 6. Conclusion

In conclusion, our approach of modular design strategy was a good solution in implementing the fire fighting robot as it made it easier for individuals to work on their tasks independently. The extensive use of microcontrollers ensured the integration step to be simpler. There were still some problems at integration step but they were solved easily because debugging can be done on each module.

Therefore, our final model of the robot can successfully find “fire” and reach it without running into obstacles. Also, we managed to construct the robot comfortably within the budget of $400.

Throughout the project, our technical knowledge was put to practical use and hence learnt many technical skills. This project also made us aware of the markets of technical components in NZ and many other factors we didn’t realise before.

## References

Custom Computer Service (CCS) Inc. (2006). *Development Kit for the PIC MCU Exercise book: PIC16F877A.*

Eltec Pyroelectric detector. (2009). Acroname Robotics. Retrieved 1st April, 2009 from http://www.acroname.com/robotics/parts/R1-442-3.html

Microchip Technology Inc. (2001). *PIC 16F87X Datasheet.*

OSRAM. (2005). *Silicon NPN Transistor (SFH309 PFA) Datasheet.* Published by OSRAM Opto Semiconductors GmbH.

RS New Zealand. (n.d.). Retrieved March 26th 2009, from http://newzealand.rs-online.com/web/home.html

Sharp Corporation. (2006). *GP2D120 Optoelectronic device datasheet*.

(2009). Video of our robot. Retrieve from http://jamesyxu.com/Robot%20Final.zip

Sharp IR Rangers Information. (2008). Acroname Robotics. Retrieved 18th May, 2009 from http://www.acroname.com/robotics/info/articles/sharp/sharp.html#e8

UVTron Flame detector. (2008). Acroname Robotics. Retrieved 1st April, 2009 from http://www.acroname.com/robotics/parts/R66-R2868.html

## Appendix A

### Main controller code

* “chipConf.h”

#include <16F877A.h>

#device adc=8

#FUSES NOWDT //No Watch Dog Timer

#FUSES HS //High speed Osc (> 4mhz)

#FUSES NOPROTECT //Code not protected from reading

#FUSES NOBROWNOUT //No brownout reset

#FUSES PUT //No Power Up Timer

#FUSES NOLVP //Low Voltage Programming on B3(PIC16) or B5(PIC18)

#use delay(clock=10000000)

#use rtos(timer=0,minor\_cycle=10ms)

* Main controller code

#include "chipConf.h"

const int8 STATUS\_PANEL\_RED = 0x00; const int8 STATUS\_PANEL\_YELLOW = 0x01; const int8 STATUS\_PANEL\_GREEN = 0x02;

const int8 PROXIMITY\_NO\_OBJECT = 0; const int8 PROXIMITY\_LEFT = 2; const int8 PROXIMITY\_RIGHT = 1; const int8 PROXIMITY\_FRONT = 3;

const int8 FLAME\_NONE = 0; const int8 FLAME\_LEFT = 1; const int8 FLAME\_RIGHT = 2; const int8 FLAME\_FRONT = 4; const int8 FLAME\_REAR = 3; const int8 FLAME\_NEAR = 5;

const int8 MOTOR\_STOP = 0; const int8 MOTOR\_LEFT = 1; const int8 MOTOR\_RIGHT = 2; const int8 MOTOR\_FORWARD = 3;

const int8 RECURSIVE\_MAX = 1;

const int8 FAN\_ON = 0;const int8 FAN\_OFF = 1;

void displayStatus(int8, int8); void fanCtrl(int8); void initAll(); void fightFire(); void stopRecursion();

//global vars int8 valA = 0; int8 valB = 0; int8 gCount = 0; int8 gCount2 = 0; int8 gMotor = 0; int8 lastDir = 0;

int8 recCount = 0;

//RTOS Prototypes #task(rate=600ms, queue=2) void motorCtrl(); #task(rate=100ms) void rtos\_CA(); #task(rate=150ms)

void rtos\_FT();

void stopRecursion() { displayStatus(STATUS\_PANEL\_YELLOW, 0x09);

rtos\_disable(rtos\_CA);

//right turn output\_high(PIN\_C1); output\_high(PIN\_C2); output\_low(PIN\_C3);

delay\_ms(3000);

rtos\_enable(rtos\_CA);

displayStatus(STATUS\_PANEL\_YELLOW, 0x00);

}

void fightFire() {

displayStatus(STATUS\_PANEL\_YELLOW, 0x01); rtos\_disable(rtos\_CA);

rtos\_disable(rtos\_FT);

displayStatus(STATUS\_PANEL\_YELLOW, 0x02);

//gMotor = MOTOR\_STOP; output\_low(PIN\_C1); output\_low(PIN\_C2); output\_low(PIN\_C3);

fanCtrl(FAN\_ON); delay\_ms(10000);

fanCtrl(FAN\_OFF);

//right turn output\_high(PIN\_C1); output\_high(PIN\_C2); output\_low(PIN\_C3);

delay\_ms(3000);

displayStatus(STATUS\_PANEL\_YELLOW, 0x00);

rtos\_enable(rtos\_CA); rtos\_enable(rtos\_FT);

}

void rtos\_FT() {

//============Start of Flame Tracking======== valA = input\_a(); valA = valA & 0x0F;

//============End of Flame Tracking==========

switch (valA) { case FLAME\_NONE: displayStatus(STATUS\_PANEL\_RED, 0x00); if (gCount2 > 0) gCount2--; else

rtos\_enable(rtos\_CA);

//gMotor = MOTOR\_FORWARD; break;

case FLAME\_LEFT: displayStatus(STATUS\_PANEL\_RED, 0x01); rtos\_disable(rtos\_CA); gCount2 = 10; gMotor = MOTOR\_LEFT; break;

case FLAME\_RIGHT: displayStatus(STATUS\_PANEL\_RED, 0x02); rtos\_disable(rtos\_CA); gCount2 = 10; gMotor = MOTOR\_RIGHT; break;

case FLAME\_REAR: displayStatus(STATUS\_PANEL\_RED, 0x04); rtos\_disable(rtos\_CA); gCount2 = 10; gMotor = MOTOR\_LEFT; break;

case FLAME\_FRONT:

displayStatus(STATUS\_PANEL\_RED, 0x08); gMotor = MOTOR\_FORWARD; gCount2 = 10; rtos\_enable(rtos\_CA); break;

case FLAME\_NEAR: displayStatus(STATUS\_PANEL\_RED, 0x0C); fightFire(); break;

default: displayStatus(STATUS\_PANEL\_RED, 0x00); break;

}

}

void rtos\_CA() {

//============Start of Collision Detection=== valB = input\_b();

while ((valB & 0x01) == 0) { delay\_ms(5); valB = input\_b();

}

valB = valB & 0x07; valB = valB >> 1;

//============End of Collision Detection=====

switch (valB) { case PROXIMITY\_NO\_OBJECT:

displayStatus(STATUS\_PANEL\_GREEN, 0x01);

if (gCount > 0) gCount--;

else

rtos\_enable(rtos\_FT); gMotor = MOTOR\_FORWARD; break;

case PROXIMITY\_LEFT: displayStatus(STATUS\_PANEL\_GREEN, 0x02); rtos\_disable(rtos\_FT); gCount = 20; gMotor = MOTOR\_RIGHT; if (lastDir == MOTOR\_LEFT)

recCount++;

if (recCount >= RECURSIVE\_MAX) {

stopRecursion();

recCount = 0;

}

lastDir = MOTOR\_RIGHT; break;

case PROXIMITY\_RIGHT: displayStatus(STATUS\_PANEL\_GREEN, 0x04); rtos\_disable(rtos\_FT); gCount = 20; gMotor = MOTOR\_LEFT; lastDir = MOTOR\_LEFT; break;

case PROXIMITY\_FRONT: displayStatus(STATUS\_PANEL\_GREEN, 0x06); rtos\_disable(rtos\_FT); gCount = 20; gMotor = MOTOR\_RIGHT; break;

default:

displayStatus(STATUS\_PANEL\_GREEN, 0x00); gMotor = MOTOR\_STOP; break;

}

}

void initAll() {

set\_tris\_a(0x1F); set\_tris\_b(0x0F); set\_tris\_c(0x00); set\_tris\_d(0x00); set\_tris\_e(0x00);

setup\_adc(ADC\_OFF);

setup\_adc\_ports(NO\_ANALOGS);

output\_c(0x00); output\_d(0x00);

output\_low(PIN\_D1);

//motorStop output\_low(PIN\_C1); output\_low(PIN\_C2); output\_low(PIN\_C3);

output\_high(PIN\_D3); delay\_ms(5000); output\_low(PIN\_D3);

}

void fanCtrl(int8 on) { if (on == FAN\_ON) output\_high(PIN\_D1); else output\_low(PIN\_D1);

}

void displayStatus(int8 panel, int8 val) {

int8 lsb = 0x00;

int8 div[4];

switch (panel) { case STATUS\_PANEL\_RED:

//Red panel has D2(Edge), D3, C4, C5 lsb = val & 0x0F; div[0] = lsb & 0x01; div[1] = lsb & 0x02; div[2] = lsb & 0x04; div[3] = lsb & 0x08; if (div[0] == 0)

output\_low(PIN\_D2); else output\_high(PIN\_D2); if (div[1] == 0)

output\_low(PIN\_D3); else

output\_high(PIN\_D3); if (div[2] == 0) output\_low(PIN\_C4); else output\_high(PIN\_C4); if (div[3] == 0) output\_low(PIN\_C5); else output\_high(PIN\_C5); break;

case STATUS\_PANEL\_YELLOW: //Yellow has C6(To red), C7, D4 lsb = val & 0x0F; div[0] = lsb & 0x01; div[1] = lsb & 0x02; div[2] = lsb & 0x04; if (div[0] == 0) output\_low(PIN\_C6); else output\_high(PIN\_C6); if (div[1] == 0) output\_low(PIN\_C7); else output\_high(PIN\_C7); if (div[2] == 0)

output\_low(PIN\_D4); else output\_high(PIN\_D4); break;

case STATUS\_PANEL\_GREEN: //Green has D5(To yellow), D6, D7 lsb = val & 0x0F; div[0] = lsb & 0x01; div[1] = lsb & 0x02; div[2] = lsb & 0x04; if (div[0] == 0)

output\_low(PIN\_D5); else output\_high(PIN\_D5); if (div[1] == 0)

output\_low(PIN\_D6); else output\_high(PIN\_D6); if (div[2] == 0)

output\_low(PIN\_D7); else output\_high(PIN\_D7); break;

}

}

void motorCtrl() {

int8 dir = 0;

/\* if(rtos\_msg\_poll() > 0) dir = rtos\_msg\_read(); else return;

\*/

dir = gMotor;

switch (dir) { case MOTOR\_STOP: output\_low(PIN\_C1); output\_low(PIN\_C2); output\_low(PIN\_C3); break;

case MOTOR\_LEFT: output\_low(PIN\_C1); output\_high(PIN\_C2); output\_low(PIN\_C3); break;

case MOTOR\_RIGHT: output\_high(PIN\_C1); output\_high(PIN\_C2); output\_low(PIN\_C3); break;

case MOTOR\_FORWARD: output\_low(PIN\_C1); output\_low(PIN\_C2); output\_high(PIN\_C3); break;

default: output\_low(PIN\_C1); output\_low(PIN\_C2); output\_low(PIN\_C3); break;

}

}

void main(){

initAll();

rtos\_run();

}

## Appendix B

### Flame detection code

#include <16F877A.h>

#DEVICE ADC=10

#fuses XT, NOLVP, NOWDT, NOPROTECT, NOBROWNOUT, PUT #use delay(clock=4000000)

#define FLAME\_NO\_SRC 0x00

#define FLAME\_SRC\_LEFT 0x01

#define FLAME\_SRC\_RIGHT 0x02

#define FLAME\_SRC\_REAR 0x03

#define FLAME\_SRC\_FRONT 0x04

#define FLAME\_SRC\_FAN 0x05

#define THRESHOLD 163 //0.8v

#define MIN\_BASE 655 //3v #define FAN\_RANGE 820 //4v

void initAdc(); void readSensors(); int16 getBaseline(); void detectSrc(int16 baseline);

void outputSrc(int8 pos);

int16 sensors[6] = {0};

int16 sortedSensors[6] = {0};

void initAdc() {

SETUP\_ADC(ADC\_CLOCK\_INTERNAL);

SETUP\_ADC\_PORTS(ALL\_ANALOG);

set\_tris\_a(0x2F); set\_tris\_b(0x00); set\_tris\_c(0x00); set\_tris\_d(0x00); set\_tris\_e(0x07);

output\_b(0x00);

}

void readSensors() {

int8 i;

for(i=0; i<6; i++) {

set\_adc\_channel(i); sensors[i] = read\_adc(); sortedSensors[i] = sensors[i]; }

}

int16 getBaseline() {

/\* procedure bubbleSort( A : list of sortable items ) defined as: do swapped := false for each i in 0 to length(A) - 2 inclusive do:

if A[ i ] > A[ i + 1 ] then

swap( A[ i ], A[ i + 1 ] ) swapped := true

end if

end for

while swapped

end procedure

\*/ int8 swapped = 0; //0 is false int8 i; int16 hold;

int16 average;

do {

swapped = 0;

for(i=0; i<=6-2; i++) {

if(sortedSensors[i] > sortedSensors[i+1]) { hold = sortedSensors[i]; sortedSensors[i] = sortedSensors[i+1]; sortedSensors[i+1] = hold; swapped = 1;

}

}

} while(swapped);

average = sortedSensors[0] + sortedSensors[1] + sortedSensors[2]; average = (average+1) / 3; //avoid division by 0 return average;

}

void detectSrc(int16 base) {

int8 i, j;

for(i=6; i>0; i--) { if ((sortedSensors[i-1] > base) && (sortedSensors[i-1] > MIN\_BASE)) {

for(j=0; j<6; j++) { if (sensors[j] == sortedSensors[i-1]) {

if ((sensors[j] >= FAN\_RANGE) && (j == 0))

outputSrc(6);

else

outputSrc(j);

return;

}

}

}

}

outputSrc(99);

}

void outputSrc(int8 pos) {

/\*

#define FLAME\_NO\_SRC 0x00

#define FLAME\_SRC\_LEFT 0x01

#define FLAME\_SRC\_RIGHT 0x02

#define FLAME\_SRC\_REAR 0x03

#define FLAME\_SRC\_FRONT 0x04

\*/

switch (pos) {

case 0:

output\_b(FLAME\_SRC\_FRONT); break; case 1: case 2:

output\_b(FLAME\_SRC\_LEFT); break; case 3: case 4:

output\_b(FLAME\_SRC\_RIGHT); break;

case 5:

output\_b(FLAME\_SRC\_REAR); break;

case 6:

output\_b(FLAME\_SRC\_FAN); break;

case 99:

output\_b(FLAME\_NO\_SRC); break;

}

}

int main() {

int16 baseline;

initAdc();

while(1) {

readSensors(); baseline = getBaseline();

detectSrc(baseline + THRESHOLD);

}

}

## Appendix C

### Motor control code

#include <16F877A.h>

#fuses XT, NOLVP, NOWDT, NOPROTECT, NOBROWNOUT, PUT #use delay(clock=4000000)

int8 B3, B2, B1;

void stop(); void left(); void right(); void forward(); void backward();

void stop() {

output\_low(PIN\_D0); output\_low(PIN\_D1); output\_low(PIN\_D2); output\_low(PIN\_D3);

}

void left() {

output\_low(PIN\_D0); output\_high(PIN\_D1); output\_high(PIN\_D2); output\_low(PIN\_D3);

}

void right() {

output\_high(PIN\_D0); output\_low(PIN\_D1); output\_low(PIN\_D2); output\_high(PIN\_D3);

}

void forward() { output\_high(PIN\_D0); output\_low(PIN\_D1); output\_high(PIN\_D2); output\_low(PIN\_D3);

}

void backward() { output\_low(PIN\_D0); output\_high(PIN\_D1); output\_low(PIN\_D2); output\_high(PIN\_D3);

}

void main() { set\_tris\_a(0); // Setting unused pins to output set\_tris\_b(0x0E); set\_tris\_c(0); set\_tris\_d(0); set\_tris\_e(0);

output\_low(PIN\_D0); output\_low(PIN\_D1); output\_low(PIN\_D2); output\_low(PIN\_D3);

while(1) {

B1=input(PIN\_B1);

B2=input(PIN\_B2);

B3=input(PIN\_B3);

if (!B1 && !B2 && !B3) stop();

else if (!B1 && B2 && !B3) left(); else if (B1 && B2 && !B3) right(); else if (!B1 && !B2 && B3) forward(); else if (B1 && !B2 && B3) backward();

}

}

## 

## Appendix D

## Proximity code

Note the functions being called in the driver’s code, are define in the PIC’s header file.

*#ifndef OBJECT\_SENSORS*

*#define OBJECT\_SENSORS*

*//defines the pin on the PIC for the left sensor*

*#ifndef LEFT\_ADC\_CHANNEL*

*#define LEFT\_ADC\_CHANNEL PIN\_A2*

*#endif*

*//defines the pin on the PIC for the right sensor*

*#ifndef RIGHT\_ADC\_CHANNEL*

*#define RIGHT\_ADC\_CHANNEL PIN\_A1*

*#endif*

*//defines the pin on the PIC for the middle sensor*

*#ifndef MID\_ADC\_CHANNEL*

*#define MID\_ADC\_CHANNEL PIN\_A3*

*#endif*

// *Purpose: Initialize the analog to digital converter*

// *Inputs: None* // *Outputs: None void init\_objectSensors()*

*{*

*SETUP\_ADC(ADC\_CLOCK\_INTERNAL);*

*SETUP\_ADC\_PORTS(ALL\_ANALOG);*

*}*

// *Purpose: Read the ADC value from the left sensor*

// *Inputs: None*

// *Outputs: The value read from the left sensor's ADC port*

*#if getenv("ADC\_RESOLUTION") == 8*

*int8 read\_leftObjectSensor()*

*#else int16 read\_leftObjectSensor()*

*#endif*

*{*

// *Set the ADC to read the left object sensor*

// *Allow the voltage to stabilize delay\_ms(1);*

// *Read the value from the left sensor*  *return READ\_ADC();*

*}*

// *Purpose: Read the ADC value from the right sensor*

// *Inputs: None*

// *Outputs: The value read from the right sensor's ADC port*

*#if getenv("ADC\_RESOLUTION") == 8*

*int8 read\_rightObjectSensor()*

*#else*

*int16 read\_rightObjectSensor()*

*#endif*

*{*

// *Set the ADC to read the right object sensor*

*SET\_ADC\_CHANNEL(RIGHT\_ADC\_CHANNEL);*

// *Allow the voltage to stabilize* *delay\_ms(1);*

// *Read the value from the right sensor*  *return READ\_ADC();*

*}*

// *Purpose: Read the ADC value from the middle sensor*

// *Inputs: None*

// *Outputs: The value read from the middle sensor's ADC port*

*#if getenv("ADC\_RESOLUTION") == 8*

*int8 read\_midObjectSensor()*

*#else int16 read\_midObjectSensor()*

*#endif*

*{*

// *Set the ADC to read the middle object sensor*

*SET\_ADC\_CHANNEL(MID\_ADC\_CHANNEL);*

// *Allow the voltage to stabilize delay\_ms(1);*

// *Read the value from the right sensor*  *return READ\_ADC();*

*}*

// *Purpose: Read the ADC values from both sensors*

// *Inputs: 1) A reference to the location to store the left measurement*

// *2) A reference to the location to store the right measurement*

// *Outputs: None*

*#if getenv("ADC\_RESOLUTION") == 8*

*void read\_objectSensors(int8& leftSensor, int8& rightSensor, int8& midSensor)*

*#else*  *void read\_objectSensors(int16& leftSensor, int16& rightSensor, int16& midSensor)*

*#endif*

*{*

// *Set the ADC to read the left object sensor*

*SET\_ADC\_CHANNEL(LEFT\_ADC\_CHANNEL);*

// *Allow the voltage to stabilize* *delay\_ms(1);*

// *Read the value from the left sensor leftSensor = READ\_ADC();*

// *Set the ADC to read the right object sensor*

*SET\_ADC\_CHANNEL(RIGHT\_ADC\_CHANNEL);*

// *Allow the voltage to stabilize* *delay\_ms(1);*

// *Read the value from the right sensor rightSensor = READ\_ADC();*

// *Set the ADC to read the middle object sensor*

*SET\_ADC\_CHANNEL(MID\_ADC\_CHANNEL);*

// *Allow the voltage to stabilize*

*delay\_ms(1);*

// *Read the value from the middle sensor*  *midSensor = READ\_ADC();*

*}*

*#endif*

The next bit of code is to define the events and output accordenly:

*#include <GP2D1203sensor.c>*

*//this defines the distance we compare with the outputs from the ADC #define PRX\_DETECT 32*

*void* *procProximity(); int8 left();* *int8 right();* *int8 center();* *int8 leftSense;* *int8 rightSense;* *int8 midSense;*

*void procProximity() {**leftSense = rightSense = midSense = 0;* *while(true) {*

*//event loop calls functions defined in the sensor driver* *read\_objectSensors(leftSense, rightSense, midSense);*

*//leftSense = midSense;* *//delay\_ms(10000);*

*if (!left() && !right() && !center()) {* *//if all sensor no detect output 00*

*OUTPUT\_LOW(PIN\_B0);*

*OUTPUT\_LOW(PIN\_B1);*

*OUTPUT\_LOW(PIN\_B2);*

*OUTPUT\_HIGH(PIN\_B0);*

*} else if ((left() && !right() && !center()) || (left() && !right() && center()))*

*{ //if left or left and center output 01(left)*

*OUTPUT\_LOW(PIN\_B0);*

*OUTPUT\_LOW(PIN\_B1);*

*OUTPUT\_HIGH(PIN\_B2);*

*OUTPUT\_HIGH(PIN\_B0);*

*} else if ((!left() && right() && !center()) || (!left() && right() && center()))*

*{ //if right or right and center output 10(right)*

*OUTPUT\_LOW(PIN\_B0);*

*OUTPUT\_HIGH(PIN\_B1);*

*OUTPUT\_LOW(PIN\_B2);*

*OUTPUT\_HIGH(PIN\_B0);*

*} else if ((!left() && !right() && center()) || (left() && right() && !center()) || (left() && right() &&*

*center())) {*

*//if center or left and right or all output*

*11(ALL) OUTPUT\_LOW(PIN\_B0);*

*OUTPUT\_HIGH(PIN\_B1);*

*OUTPUT\_HIGH(PIN\_B2);*

*OUTPUT\_HIGH(PIN\_B0);*

*}*

*}*

*}*

*int8* *left() {* *//return 0;* *if (leftSense <= PRX\_DETECT)* *return 0;* *else* *return 1;*

*}*

*int8 right() {*

*if (rightSense <= PRX\_DETECT)* *return 0;* *else* *return 1;*

*}*

*int8 center() {**//return 0;* *if (midSense <= PRX\_DETECT)* *return 0;* *else* *return 1;*

*}*

This is then used in the main program:

#include <16F877A.h>

#fuses XT,NOLVP,NOWDT,NOPROTECT,NOBROWNOUT,PUT

#use delay(clock=4000000) #include "proximityControl2.c" void main() { int8 leftSense; int8 rightSense; int8 midSense; init\_objectSensors(); procProximity();

}