Autonomous Vehicle Challenge Report

Abstract

This report gives a detailed explanation for the process of designing and building an autonomous vehicle as a team with the aim of successfully navigating (from start to finish) a maze in a competition. The robot to managed go through the maze in the allotted time with some minor hiccups. This success is due, in part, to the efficient design and implementation (code) of the robot but most importantly due to the undeterred commitment of each team to finish what each one of us had started.

Introduction

This report gives a detailed explanation for the process of designing and building an autonomous vehicle as a team with the aim of successfully navigating (from start to finish) a maze in a competition. The maze is split into four quadrants:

Quadrant	**Unique Feature**	**Points Value (out of 100%)**
Alpha	Wiggly white line on black background	20%
Beta	White line arranged in maze pattern	20%
Gamma	Same as Beta quadrant but with some added obstacles that interrupt the guide line	20%
Delta	Guiding white line can have breaks in it. Vertical walls can be added in tunnel fashion	20%
Additional Requirements		
Accurately reporting robot position (quadrant) over wireless link to the base-station		20%

Each team is given 15 minutes to complete this task. In the case where a robot gets stuck in a quadrant e.g. after bumping a wall, the maximum grade for a team will be noted. A team may choose to re-do the maze from the start if they wish (provided there is still time). Aside from these, there are additional mandatory requirements. These are:

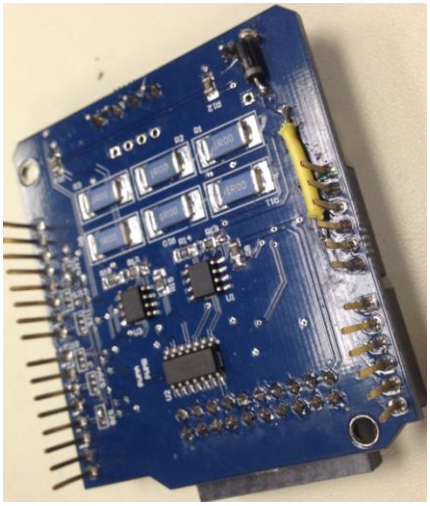
Additional Mandatory Requirements	**Maximum Possible Point Deduction**
Recyclability – how easy is it to get the parts back (easy to assembled and detach) Also making it recyclable makes it easier to repair or change if you discover a problem with current design.	-10% to team score
Aesthetics – how good it looks (no hanging wires from robot) Does it look like a mess of wires and parts or is it tidy, or colourful etc.	-10% to team score
Weekly Blog – maintain brief progress reports through team blog This also helps to keep project on track	-5% to individual score
Vehicle Access – team members may take their vehicle out of CO145 to take it up to CO242a or CO242b upon one condition, they will be required to sign out the vehicle out of CO145 when they leave with the vehicle, and must sign back in when they bring it back to CO145.	-5% to team score

Background

Arduino

Arduino is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. It can be used to develop interactive objects, taking inputs from a

variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. It can be programmed using an open-source IDE, based on the Processing multimedia programming environment (similar to C).[1]



Shields

Shields (shown left) are boards that can be plugged on top of the Arduino PCB extending its capabilities.[2]

Sensors

QTR-8RC Reflectance Sensor

The QTR-8RC sensor from Pololu works using infrared emitters and receiver. It uses an interesting method to extract analogue data about what it sees via the use of digital pins. The sensor comprises of 8 individual sensor elements, each containing an emitter/receiver, and a capacitor. The sensor works by having the micro controller set each I/O pin to high, which charges the capacitor for each element slightly.

Distance Sensors

Short Range

The Sharp 5cm Digital Distance sensor from Pololu works using infrared and it detects objects between 0.5 and 5 cm (0.2" and 2") away. With its quick response time, small size, low current draw, and short minimum sensing distance, this sensor is a good choice for non-contact and close-proximity object detection.[3] The output, Vo, is driven low when the sensor detects an object; otherwise, the output is high.

Long Range

The Sharp 4 cm to 30 cm Analogue Distance sensor from Pololu works using infrared as well and it detects objects between 1.5" to 12" away. This is more economical than sonar rangefinders, yet it provides much better performance than other IR alternatives. Interfacing to most microcontrollers is straightforward: the single analog output can be connected to an analog-to-digital converter for taking distance measurements.[4]

Xbee Radio Transceiver

The Arduino Xbee shield allows multiple Arduino boards to communicate wirelessly over distances up to 100 feet (indoors) or 300 feet (outdoors) using the Maxstream Xbee Zigbee module. It can be used as a serial/usb replacement or you can put it into a command mode and configure it for a variety of broadcast and mesh networking options.[5]

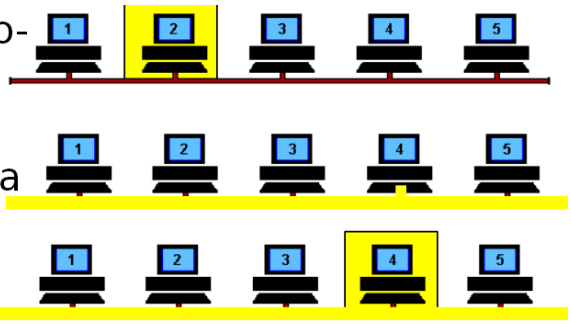
Ethernet Frames[6]

- Data is sent as “frames”.
- Each frame has:
    - Destination address
    - Source address (so base station knows who to return acknowledgements to, if necessary).
    - Data (up to 1500 bytes)
- The addresses are Media Access Control addresses (MACs), always six bytes
- Usually assigned to the Ethernet adapter and globally unique

---

[1] Arduino.cc, 2014
[2] Arduino.cc, 2014
[3] Pololu.com, 2014
[4] Pololu.com, 2014
[5] Arduino.cc, 2014
[6] Bryan Ng, Victoria University of Wellington – Networking (ENGR 101)

- Xbee Radio listens to see if no-one else is sending.
- Xbee Radio sends signal representing a packet of data (destination, sender, data).
- Everyone sees it but everyone ignores except Base Radio .



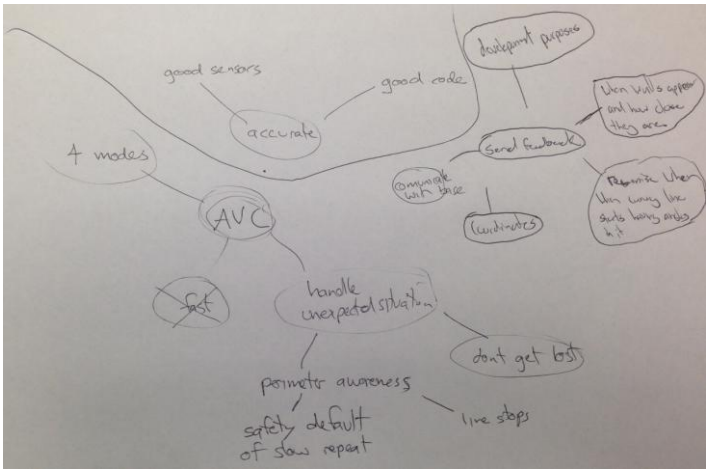
Method

In the first week of the AVC, we are asked to produce a project timeline using a Gantt chart (see below) before we can proceed with designing our robot. We divided the task into five main tasks:

1. Planning
2. Initial Design
3. Integration and Debugging
4. Networking
5. Final Testing

Each main task has sub-tasks that are done to ensure that the goals in each of the main tasks are completed. It is worthy to note though that testing and debugging is done *iteratively* whenever modifications (to the code or the robot itself) are made.

	Week 7	Holidays	Week 8	Week 9	Week 10	Week 11
Main Task 1 (Planning)						
Choose Sensors	All		All	All	All	All
Design Robot	All					
Design Gantt Chart	All		All	All	All	All
Blog Entry	All		All	All	All	All
Brainstorm Ideas	All					
Main Task 2 (Initial Design)						
Laser Cut Parts/3D Print						
Create Code For Movement						
Create Code For Collision						
Main Task 3 (Integration and Debugging)						
Debug the minor errors in the code						
Put together the robot						
Main Task 4 (Networking)						
Code for communicating with base station						
What to display?						
Main Task 5 (Testing)						
Run robot through course and refine						All



We brainstormed our ideas and the key factors that are critical to the successful completion of this team project. Some of these factors were:

- Effective sensing (stay on track, when to stop)
- Efficient code
- Adaptation (handle unexpected situations)
- Move fast
- Debugging methods (how we test a machinery that has some degree of complication)
- Effective communication of quadrant location to base (tracking using some sort of coordinate

location)

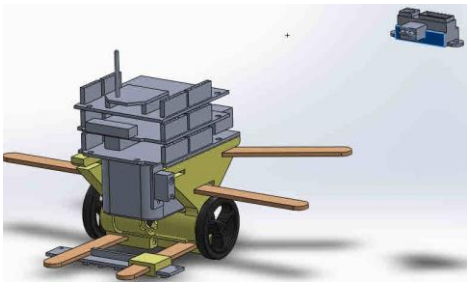
Some of these factors needed to traded-off for sake of accurate sensing such as the speed and the placement of sensors. It's going to be difficult to make fast and accurate robot at the same time. In doing the Gantt chart and this mind-map, we have discovered something that is very important in any team project – that the tasks needed to be broken down and allocated to team members. Our earlier approach which is one or two people coding and the others staring them is evidently not very effective. With the limited timeframe (5 weeks + Easter break) given, we need to make sure that we don't waste valuable time. We divided the team into subgroups/'sub-teams' and each team member's allocation is seen in the table on the right while the tasks that each group needed to complete is given in the table below.

People	
William Thomson	
Ronni Perez	
Reuben Puketapu	
Jelle van der Have	
Daniel Ekers	
Kieran van de Riet	
All	
Group	
Software Team	
Daniel Ekers	
Jelle van der Have	
Hardware Team	
William Thomson	
Reuben Puketapu	
Network Team	
Ronni Perez	
Kieran van de Riet	

Group	**Areas of Concern**
Hardware Team	- Formulate 3D models of robot using CAD software based on agreed team design - 3D print parts and laser cut them (if necessary) - Assemble the robot - Refine the robot design (wheels, chassis) or sensor placements (if necessary) upon instructions of the software team
Software Team	- Create code for movement o Motor speed control o Line detection and following o 180^{O} turn upon detecting line break - Create code for collision o Wall detection and avoidance
Network Team	- Send data from robot to base station o Display quadrant number and team name o Can be used for debugging – display sensor readings - Integrate network code to main code made by software team

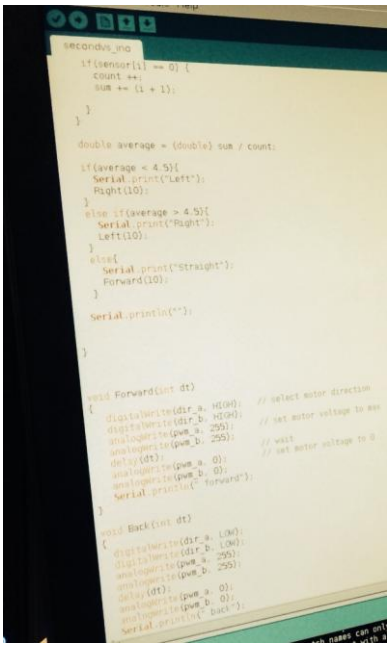
We did this because this methodology is proven to be effective and practical. It is used especially in large software development companies such as Google.

At the beginning of this project (during the Easter break + Week 8), the hardware team had decided to make a prototype/test rig (see right) so we could come up with our final design by testing different sensor locations, different motor-control coding methods and adjusting necessary delay times within code execution. We did this because when we used electrical tape to hold everything together (in Week 7) so we could test the code, it was really difficult to move a sensor to different locations in the robot e.g. lifting the QTR line sensor a few millimetres up so it can actually detect light.



We have used the provided lineSensor code and motor code as the base code for the prototype. The provided base code can:

- Read sensor values from QTR line sensor emitters and outputs them as integers
- Control motor speed (rotations/time) using pulse-width modulation
- Control direction of vehicle (turning motor on/off)

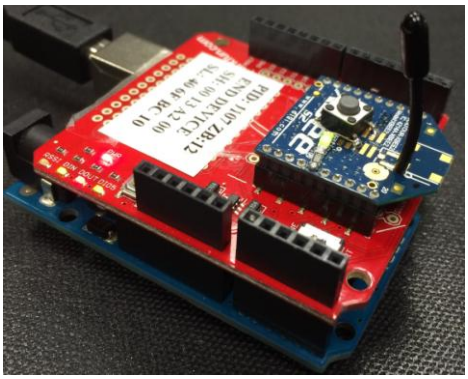


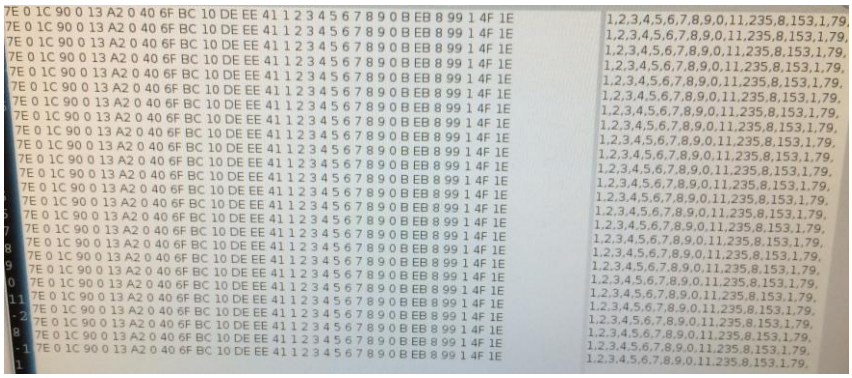
The software team have decided to reduce noise by 'digitising' QTR line sensor readings – that is if a particular sensor reading is more than 100 we would set it to 1, if not set it to 0 (see left). We then took the average (mean) of all sensor readings and if average is 4.5 (median of 8 emitters), it means robot is in the centre of the line. We have used this coding technique for quadrants alpha to gamma.

During the Easter break, the network team began development of the network component of the robot (additional requirements). The base code was written but had no functionality as there were still a number of tweaks to be made to the existing code. The packet sent contains:

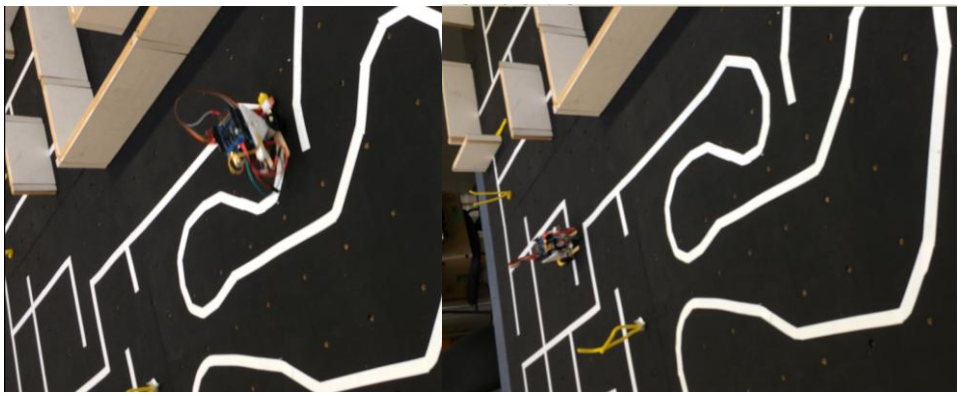
- Packet delimiter
- Length of packet
- Destination (base radio) address – both 16 bit and 64 bit
- Number of hops
- Actual Data

In Week 9, the network team spent the lab session with the goal of setting up communication with the host machine in the lab room. We did encounter multiple problems during that lab session. Problems like having a faulty Arduino board when trying to test our code but the problem that hindered our progress the most was the fault of the program on the base machine (IPHost Network Monitor) that would display the packets that it had received. The program had crashed without notice and it took a while for one of us to realise what had happened. After that misfortune, we successfully managed to send packets (numbers from 1 to 9) to the host (see below).





The software team managed to work on the maze following algorithm and the hardware team added a ShortRange IR sensor to the robot so that it could successfully complete the alpha, beta and gamma quadrants.



They started out by doing a timed turn which seemed to work quite well for both left and right. But they found the robot usually overturned (see right) and had to correct itself after each turn. This is because the robot gathers a decent amount of inertia and furthermore, by making it faster this will increase. We will have to trade-off speed with accuracy or better yet, find a way so that this momentum/inertia does not affect its turning.

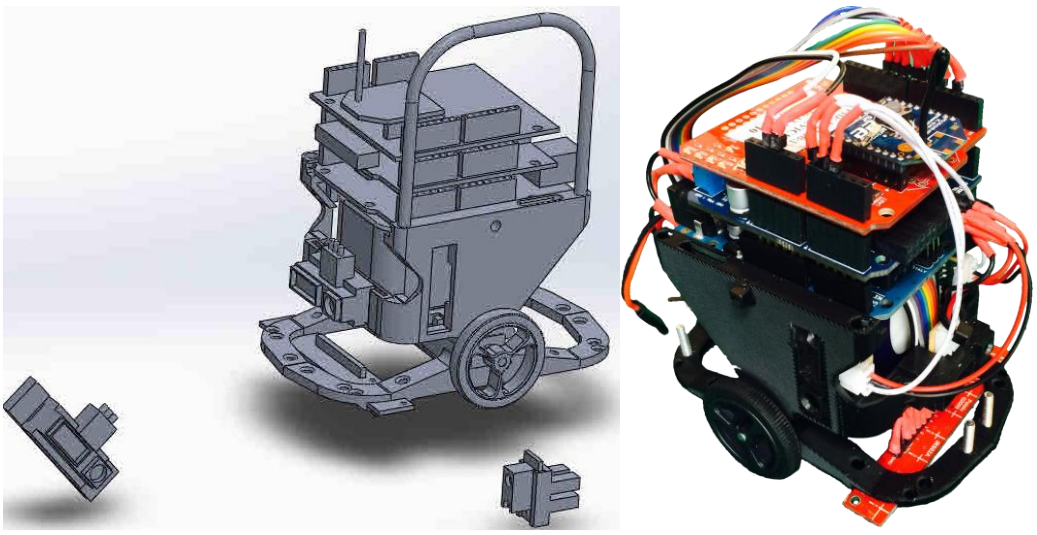
Instead of just making the turning time less we decided it would be "smarter" for the robot to just keep turning until it found a line again. This would ensure a perfect turn every time.

In Week 10, the hardware team ran into a major problem when the battery was plugged in with incorrect polarity. This caused the Arduino's Vicmoto Board to be fried and then was useless and the whole board didn't work. The damage caused the motors to run extremely slow and became inconsistent with our code. Further repairs concluded unsuccessful as only one motor worked even though both motors were correctly connected so we had to change the board as a whole. When the board was changed the robot then correctly functioned. As a consequence of this, light-emitting diodes have now been placed on the board so the same problem cannot happen again.

By this time, long-range IR sensors have been attached onto the sides and physical measurements have been taken so software team can now code the robot to carry forward through delta quadrant. We have decided to use long-range IR sensors (for the front, left and right sides) as they output analogue signals which is more accurate compared to short-range IR sensors which output digital signals (0 or 1) and we obviously would be able to detect longer distances but we have to be cautious in our code as to avoid false-positive readings.

In this same week, the network team had this idea of using booleans as it is our view that it would be easier for the integration of the two portions of code - where the software team sets up a boolean in their code (appropriately) and then call this network (communication) code that we made passing the quadrant number as an argument, having conditionals for them inside the method.

The network team did encounter a problem which puzzled group as a whole and made us rethink the whole (message relay) portion of the network code again. The data part of the packet that we are sending is outputting ambiguous characters on the screen. We questioned the functionality of our code, if it really does what it's supposed to do. The network team later found out that these ambiguous characters are representations of null elements in the packet that we are sending -- our code is actually functional and working. Fortunately, we have our working code saved from the previous week and we used that as the basis for modification.



In Week 11, the hardware team managed to complete and print-off the final CAD robot design and assemble the robot (see above). By now, the robot can successfully go through the alpha, beta and gamma quadrants. To get through the delta quadrant, the software team decided to adapt a proportional-integral-derivative (PID) mechanism for the turns (whether 180^{O} or 90^{O}). A PID mechanism calculates an error value as the difference between a measured distance between walls and a desired distance between walls. This mechanism attempts to minimize the error by adjusting the process through use of a manipulated variable (a scale factor in this case).

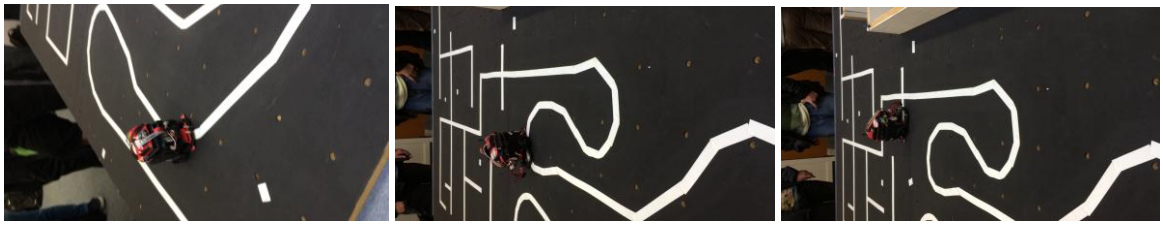
For delta quadrant, here is the idea:

- If robot detects walls:

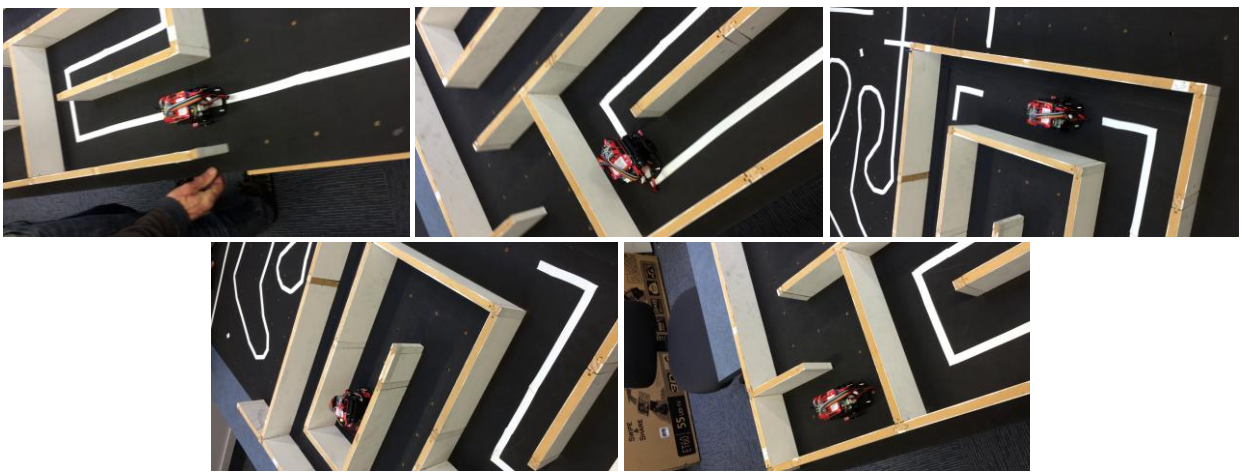
- If way in front is clear (>100mm):
  - Hug left wall (actualDistanceToLeftWall=60mm) and keep going straight (equal left and right motor speeds), provided that
- If dead end (distLeft < 80mm & distRight < 80mm & distFront < 100mm):
  - Do a 180^{O} turn
  - Stop both motors and pause for 0.5 second
- If not centred (desiredDistRight > desiredDistLeft) – happen during 90^{O} turns:
  - Do turn
  - Hug left wall (or vice-versa if no wall on other side)

At the end of Week 11, the maze code made by software team is successfully merged to the base communication code made by the network team. The group, as a whole decided not to use booleans in the base communication code as this would be inefficient (there is already a similar code for the conditionals in the maze code). We decided to use these conditional and set (char) quad to the current quadrant number (1, 2, 3 or 4) based on the maze code used.

Results



The robot coasted quadrants alpha and beta (see above) one and a half minute during the first try of the challenge but it took the robot almost nine and a half minutes (in 3 tries) to successfully complete the course as the right wall sensor was 'accidentally!?' plugged to the wrong pin on the board (see left) so the left sensor detects a wall while the right sensor throws off a random number and it detects 'another wall' at the end of gamma quadrant while actually there is none. After this has fixed by putting the right sensor connection in the right pin, the robot coasted through the entire course (see below) in the third try in roughly two and a half minutes. Our team got a perfect score.



Discussion

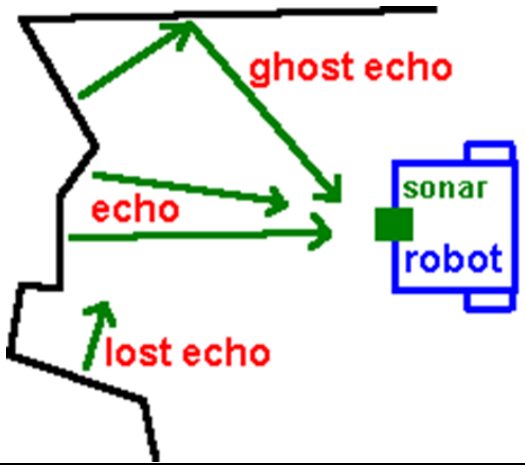
What is PID? Why?

The PID controller algorithm used for delta quadrant involves three separate constant parameters. These values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of the robot.

Advantage - Fast and accurate (it also allows many processes to run at once with no offset and low overshoot)	**Disadvantage -** Difficult to configure and tune (must have prior knowledge of measurements and constant values)

Ultrasonic vs Infrared Sensing

An ultrasonic sensor uses the speed that sound waves travel to measure distance to an object. The sensor has two openings on its front (one for emitting ultrasonic waves and the other for receiving them). The ultrasonic sensor measures distance by timing how long it takes an ultrasonic wave sent out by the emitter to bounce off an object and come back to the receiver.

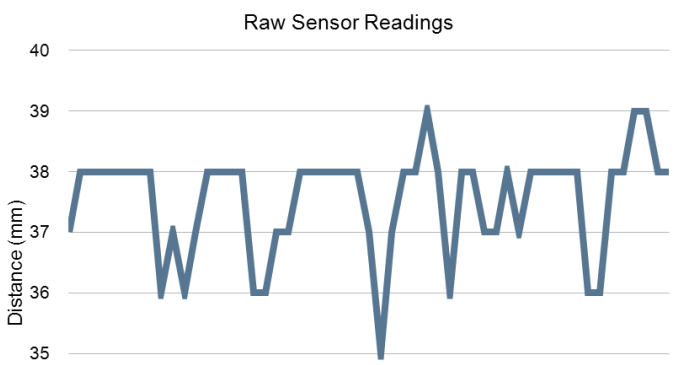
Advantage - An ultrasonic sensor's response is not dependent upon the surface color or optical (visual) reflectivity of the object.	**Disadvantage -** In reflecting a sound wave, certain materials may not reflect as well depending on the material's density, shape (see below) and ability to absorb sound 

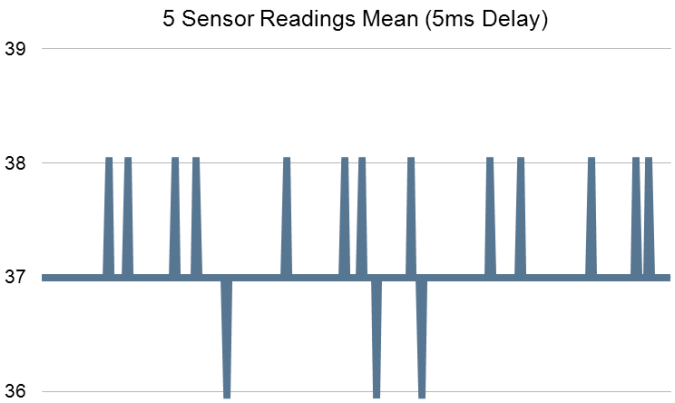
An infrared sensor, on the other hand, emits a beam of infrared light and detects reflection. Then it measures angle of reflection, not time. It outputs an appropriate voltage to represent the distance. An infrared sensor is precise at detecting objects but it is susceptible to changes in lighting (environment influence).[7]

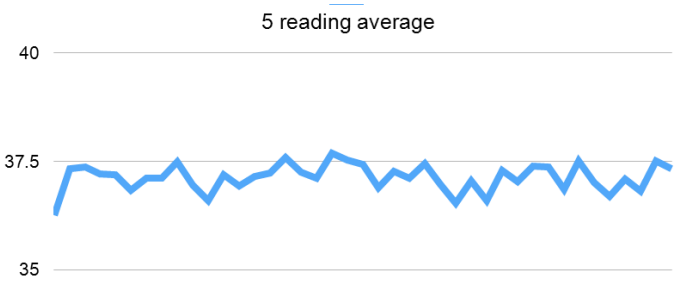
When to use…	
<u>Ultrasonic Sensors</u>	<u>Infrared Sensors</u>
- You care about distance accuracy - There will be changes in ambient lighting	- You want to detect movement or thresholds - You need precision in what you are detecting

[7] Zhang, K., Peil, J. and Kalaj, K. (2014)

Digital vs Analogue Sensor Outputs (Long-Range IR Sensors)







As seen in raw analogue sensor readings, the noise is noticeable. This would be not an accurate indication of wall distance especially for delta quadrant. Filtering (smoothing) can help to improve measurement accuracy.

This is why the group decided to filter the sensor readings by taking 5 consecutive sensor readings, taking their mean and feeding them into the PID mechanism.

Conclusion

The Autonomous Vehicle Challenge presented itself as a difficult task even for a team which consisted of 6 members. Our team was able to achieve its main goal of making an autonomous vehicle that is capable of tackling a maze problem divided into four sections (quadrants), each with increasing difficulty. Our team also achieved the additional requirements such as the communication of quadrant location to base and the robot was both aesthetically pleasing and recyclable as we did not get any deductions from our final score.

During this process, our team has had to make and ponder with different physical design decisions and coding design decisions. We encountered problems especially during the testing and debugging process but these problems were overcome by working in the spirit of teamwork – combining different coding techniques whilst still achieving one goal. We ensured that each team member contributes at least a certain amount of effort by dividing the group in sub-teams.

References

Arduino.cc, (2014). *Arduino - Introduction*. [online] Available at: http://arduino.cc/en/Guide/Introduction [Accessed 31 May. 2014].

Pololu.com, (2014). *Pololu Carrier with Sharp GP2Y0D805Z0F Digital Distance Sensor 5cm*. [online] Available at: http://www.pololu.com/product/1132 [Accessed 31 May. 2014].

Pololu.com, (2014). *Pololu - Sharp GP2D120XJ00F Analog Distance Sensor 4-30cm*. [online] Available at: http://www.pololu.com/product/1136 [Accessed 1 Jun. 2014]

Zhang, K., Peil, J. and Kalaj, K. (2014). *Distance Sensors*.

Appendix

Final Code

```
#include <QTRSensors.h>
#define NUM_SENSORS 8   // constant value
QTRSensorsRC qtr((unsigned char[]) {
2,4,5,6,7,8,9,10}, NUM_SENSORS);

float KP = 0.1; //floating-point proportional constant
float KD = 1; //floating-point derivative constant
float wallKP = 0.1;
float wallKD = 1;
int M1 = 100; //base motor speed
int M2 = 100; //base motor speed
int maxMotorValue = 255; //maximum motor speed
float motorBalance=1;
int lastWallError = 0;
int idealDistanceToTheWall = 50; //mm

unsigned int sensors[8];
unsigned int sensorsSimple[8];

int rightSpeed = 3;  //PWM control for motor outputs
1 and 2 is on digital pin 3
int leftSpeed = 11;  //PWM control for motor outputs
3 and 4 is on digital pin 11
int dirRight = 12;  //dir control for motor outputs 1
and 2 is on digital pin 12
int dirLeft = 13;  //dir control for motor outputs 3 and
4 is on digital pin 13

int lastError = 0; //Variable to store the last error

int count;
double average;

int loopCount = 0;
char quad = '1';

void setup()
{
  // optional: wait for some input from the user, such
as  a button press
  // optional: signal that the calibration phase is now
over and wait for further
  Serial.begin(9600);
  // input from the user, such as a button press
  digitalWrite(dirLeft, HIGH);   // select motor direction
  digitalWrite(dirRight, LOW);   // in this case both
forward
  analogWrite(leftSpeed, 40);
  analogWrite(rightSpeed, 40);
  // then start calibration phase and move the sensors
over both
  // reflectance extremes they will encounter in your
application:
  int i;
  for (i = 0; i < 40; i++)  // make the calibration take
about 5 seconds
  {
    qtr.calibrate();
    delay(20);
  }

  // optional: signal that the calibration phase is now
over and wait for further
  // input from the user, such as a button press
  Stop();
  Serial.println("Ready");
}

void loop()
{
  long distFront = 0;
  long distLeft = 0;
  long distRight = 0;

  //Gather the sensonr values 5 times and find
average. Increases accuracy and filters noise.
  for (int i = 0; i < 5; i++){
    distFront +=
43325*pow((double)(analogRead(A0)),-1.1129);
    distLeft += 43325*pow((double)(analogRead(A1)),-
1.1129);
    distRight +=
43325*pow((double)(analogRead(A2)),-1.1129);
    //Possible delay to increase accuracy
    delay(5);

  }

  distFront = abs(distFront/5);
  distLeft = abs(distLeft/5);
  distRight = abs(distRight/5);

  int position = qtr.readLine(sensors,
QTR_EMITTERS_ON, false);

  findAverage();
  int error = position - 3500;

  int motorSpeed = KP * error + KD * (error - lastError);
  lastError = error;

  int numberOfSensorsCovered = count;
  if (numberOfSensorsCovered == 0){
    numberOfSensorsCovered = 2;
  }
  int m1Speed = M1/numberOfSensorsCovered +
motorSpeed;
  int m2Speed = M2/numberOfSensorsCovered -
motorSpeed;
  if (m1Speed < 0) m1Speed = 0;
  if (m2Speed < 0) m2Speed = 0;

//Added This, Trying to make it go to quad 2 after
quad 1
```

```
  if((sensorsSimple[0] == 0) && (sensorsSimple[1] == 0)
&&(sensorsSimple[6]==0)&&(sensorsSimple[7]==0)&&
quad!='3'){
   //edit: stops it going back to quad 2 after its in quad
3 or 4
   if (quad='1') quad = '2';
  }

  if (distLeft < 100 && dis tRight < 100){
   quad = '4';
  }

  if (quad == '4') {

   if (distFront < 100){
    Serial.println("Time to turn");
    //Check for wall on either side and either do a 90
degree or 180 turn.
    if (distLeft < 80 && distRight < 80){
     wall180();
     Stop();
     delay(500);
    }
    else if (distRight > distLeft){
     wallRight();
     wallForward(600);
    }
    else if (distLeft > distRight){
     wallLeft();
     wallForward(600);
    }
   }

   else if (distLeft > 250){
    //wallForward(300);
    wallLeft();
    wallForward(500);
   }

   //Otherwise hug the left wall with ideal distance of
about 60mm.
   else {
    int actualDistanceToTheWall = distLeft;

    if (actualDistanceToTheWall > 60)
actualDistanceToTheWall = 60 +
((actualDistanceToTheWall - 60)/10);
    double scaleFactor = 0.008;
    double wallError = (idealDistanceToTheWall -
actualDistanceToTheWall) / scaleFactor;

    int motorSpeed = wallKP * wallError + wallKD *
(wallError - lastWallError);
    lastWallError = wallError;

    int m1Speed = 40 - (motorSpeed / 10);
    int m2Speed = 40 + (motorSpeed / 10);
    if (m1Speed < 0) m1Speed = 0;
    if (m2Speed < 0) m2Speed = 0;
    if (m1Speed > 255) m1Speed = 255;
    if (m2Speed > 255) m2Speed = 255;

    // set motor speeds using the two motor speed
variables above
    digitalWrite(dirLeft, HIGH);
    digitalWrite(dirRight, HIGH);
    analogWrite(leftSpeed, m2Speed);
    analogWrite(rightSpeed, m1Speed);

   }

  }

  //Check to see if way in front of robot is clear. In mm
  else if(distFront < 50){
   quad = '3';
   TurnAround();
   delay(400);
   qtr.readLine(sensors, QTR_EMITTERS_ON, false);
   findAverage();
   while(average != 4.5){
    qtr.readLine(sensors, QTR_EMITTERS_ON, false);
    findAverage();
   }
   Stop();
  }


  else if(isLeft()){
   digitalWrite(dirLeft, LOW);
   analogWrite(leftSpeed, 30);
   analogWrite(rightSpeed, 80);
   delay(300);
   qtr.readLine(sensors, QTR_EMITTERS_ON, false);
   findAverage();
   while(average != 4.5){
    qtr.readLine(sensors, QTR_EMITTERS_ON, false);
    findAverage();
   }
   digitalWrite(dirLeft, HIGH);
  }

  //Gone off the end of the line
  else if (count == 0){
   TurnAround();
   while(average != 4.5){
    qtr.readLine(sensors, QTR_EMITTERS_ON, false);
    findAverage();
   }
   Stop();
  }
  else{
```

```
   // set motor speeds using the two motor speed
variables above
   analogWrite(leftSpeed, m2Speed);
   analogWrite(rightSpeed,
(int)motorBalance*m1Speed);
  }
   loopCount++;
  if (loopCount = 20){
   SendToZB(quad);
   loopCount = 0;
  }
}

//Timed 180 degree turn
void wall180(){
 Serial.println("Wall 180");
 digitalWrite(dirLeft, HIGH);
 digitalWrite(dirRight, LOW);
 analogWrite(leftSpeed, 70);
 analogWrite(rightSpeed, 70);
 //delay
 delay(600);
 //digitalWrite(dirRight, HIGH);
}

//Time 90 degree left turn
void wallLeft(){
 Serial.println("Wall Left");
 digitalWrite(dirLeft, LOW);
 digitalWrite(dirRight, HIGH);
 analogWrite(leftSpeed, 0);
 analogWrite(rightSpeed, 100);
 delay(480);
 //digitalWrite(dirRight, HIGH);
}

//Time 90 degree right turn
void wallRight(){
 Serial.println("Wall Right");
 digitalWrite(dirLeft, HIGH);
 digitalWrite(dirRight, LOW);
 analogWrite(leftSpeed, 100);
 analogWrite(rightSpeed, 0);
 delay(480);
 //digitalWrite(dirRight, HIGH);
}

void wallForward(int delayTime){
 digitalWrite(dirLeft, HIGH);
 digitalWrite(dirRight, HIGH);
 analogWrite(leftSpeed, 50);
 analogWrite(rightSpeed, 50);
 delay(delayTime);
}

void TurnAround(){
 digitalWrite(dirLeft, HIGH);
 digitalWrite(dirRight, LOW);
 analogWrite(leftSpeed, 40);
 analogWrite(rightSpeed, 40);
}

void Stop(){
 digitalWrite(dirLeft, HIGH);
 digitalWrite(dirRight, HIGH);
 analogWrite(leftSpeed, 0);
 analogWrite(rightSpeed, 0);
}

boolean isLeft(){
 if((sensorsSimple[0] == 0) && (sensorsSimple[1] ==
0)) return true;
 return false;
}

int findAverage(){
 count = 0;
 for (int i = 0 ; i < NUM_SENSORS ;i++){
  if(sensors[i] > 100){
   sensorsSimple[i] = 1;
  }
  else{
   sensorsSimple[i] = 0;
  }
 }

 int sum = 0;
 for (int i = 0; i < NUM_SENSORS; i++){
  if(sensorsSimple[i] == 0) {
   count ++;
   sum += (i + 1);
  }
 }
 if (count == 0){
  average = 0;
 }
 else{
  average = (double) sum / count;
 }
}

void SendToZB(char quad)
{
 char frame_to_ZB[64];
 char data_to_send [16];
 int n_frame_to_ZB;
 int j;
 unsigned int check_sum_total = 0;
 unsigned int CRC = 0;

 //delimiter
 frame_to_ZB[0] = 0x7E;
 // length of array
 frame_to_ZB[1] = 0x00;
```

```
  frame_to_ZB[2] = 0x1E; //length                                    }
  //API ID                                                          }
  frame_to_ZB[3] = 0x10; // transmission frame
  frame_to_ZB[4] = 0x01;
  // destination 64 bit address - broadcast
  frame_to_ZB[5] = 0x00;
  frame_to_ZB[6] = 0x13;
  frame_to_ZB[7] = 0xA2;
  frame_to_ZB[8] = 0x00;
  frame_to_ZB[9] = 0x40;
  frame_to_ZB[10] = 0x6A;
  frame_to_ZB[11] = 0x40;
  frame_to_ZB[12] = 0xA4;
  // destination 16 bit address - broadcast
  frame_to_ZB[13] = 0xFF;
  frame_to_ZB[14] = 0xFE;
  // number of hops
  frame_to_ZB[15] = 0x00;
  // option
  frame_to_ZB[16] = 0x01;

  // data
  data_to_send[0] = 'q';
  data_to_send[1] = 'u';
  data_to_send[2] = 'a';
  data_to_send[3] = 'd';
  data_to_send[4] = 'r';
  data_to_send[5] = 'a';
  data_to_send[6] = 'n';
  data_to_send[7] = 't';
  data_to_send[8] = quad;
  data_to_send[9] = '-';
  data_to_send[10] = 'G';
  data_to_send[11] = 'R';
  data_to_send[12] = 'O';
  data_to_send[13] = 'U';
  data_to_send[14] = 'P';
  data_to_send[15] = '3';

  for (j=17; j < 33; j++)
  {
    frame_to_ZB[j] = data_to_send[j - 17]; // data to
send
  }
  n_frame_to_ZB = 33;

  // add CRC
  for ( j = 3 ; j < n_frame_to_ZB ; j++) {
    check_sum_total += frame_to_ZB[j];
  }
  check_sum_total = check_sum_total & 0xFF;
  CRC = 0xFF - check_sum_total;
  frame_to_ZB[n_frame_to_ZB] = CRC;
  n_frame_to_ZB += 1;

  for (j =0; j < n_frame_to_ZB; j++) {
    Serial.write(frame_to_ZB[j]);
```