SEM433 Mechatronic Design: Final Report

Dillon MacEwan

211285035

1/06/2014

Introduction

Mechatronic design incorporates skills from three traditional engineering disciplines, namely Electronic Engineering, Mechanical Engineering and Computer Science. It involves the design and interfacing of mechanical systems interfacing with the real world through sensory inputs and processed and controlled using embedded systems and programmed microcontrollers. They often have to be able to deal with variable and unpredictable environmental condition in real time, and, in order to meet cost effectiveness and robustness requirements, have to do so in a way that is computationally economic.

The task set for this report is an ideal one to test all of these principles, namely, to build an autonomous robot to complete some pre-specified tasks, and then to compete against other autonomous robots, with strict environmental, economic and physical constraints put on the project. The robotic platform has to be able to sense and interact with both static and unpredictable dynamic elements of its environment, and to be able to react instantaneously to environmental inputs, and, in the final task, to be able to react to and outperform another autonomous agent that is targeting the unit itself.

Problem analysis

The problem set in this assignment is to build an autonomous robotic unit that can navigate around a predefined arena space, and fulfil a number of set tasks. Task one involves locating a grey box, and removing it from the arena, in under three minutes. In task two, the grey box is itself an autonomous agent, attacking the robotic unit – the robotic unit must evade the attacking box for one minute. In task three, the unit has to compete against other student built units. The competition is a knockout competition consisting of 3 matches, each up to 3 minutes long. In all three tasks, if the unit or any part of the unit leaves the arena area, the task is failed or the competition lost.

The Arena is a flat, smooth circular area of 1540mm diameter. The surface will be matt black with a 20mm white stripe around the circumference and will be raised 10mm off floor level. The Sumobot must fit inside a volumetric area of 200mm width x 200mm breadth x 300mm height and weigh no more than 1kg (1000g). The Sumobot must be entirely autonomous, and is not allowed to be touched by the student after each task has commenced, and until each task is completed or failed. The Sumobot should have status indicators, be custom built by the student, and has a budgetary constraint of $150, excluding the cost of spar batteries and the microcontroller, but including all other components and parts. No offensive weaponry is permitted and overt damage to opponents Sumobots will result in disqualification – a lifting mechanism is permitted however. The box mentioned for task one and task two will by 200mm 3 and weigh 500g (Deakin University 2014).

There are a number of problems to be solved in this task then, from mechanical requirements to electrical design to programmable functionality.

The Sumobot will have to move and steer, otherwise it will not be able to even start the first task, it therefore needs motor driven wheels or tracks. Legs are overly complicated and impractical for this project, a hover bot would have no traction and find it hard to push or resist the pushing of other sumobots. A flying bot capable of catching and lifting an opponent out of the arena would be a novel approach, and the rules are non-specific about such an approach, but is perhaps a little overly complicated and hard to achieve within the allowed budget. So motorised tracks or wheels seems the most practical and sensible solution to this problem.

The motors must have ample torque to both drive the 1kg sumobot and push an opponent of equal weight, with additional force needed to overcome friction and a possible pushing force from the opposing sumobot. Ideally the toque of the motors should deliver a force at the wheel's point of contact with the floor that is just below that required to over-come the static friction force of the wheels and the floor, in order to deliver maximum acceleration without any wheel slippage. The motors and gearbox should provide enough angular velocity so that the sumobot can perform competitively, is able to outrun the automated grey box, and evade opponents if necessary, but not so fast that it can't effectively control its own motion. This will be a particular consideration in the choice of motor if a DC gearmotor is the motor of choice. Stepper Motors, DC Motors and Servo motors are all possible options for driving. Stepper motors could be problematic in needing extra software and hardware for driving, and potentially have extra cost for not much extra advantage. Servo motors will provide simplicity in control software and hardware, but may not be able to provide the torques required. DC motors will likely be cost effective, and can be controlled with PWM through an H-Bridge, but generally operate at very high angular velocities, so will need to run through a reduction gearbox, leading to extra weight and extra mechanical complexity in the sumobot. The wheels or tracks should provide enough traction to drive the sumobot without slippage, and to resist the pushing force of an opponent sumobot. Omniwheels would also be problematic in that it would be hard to provide ample resistance to the pushing force from an opposing sumobot. The weight of the motors and gearbox must be suitable so as not to use up too much of the sumobots weight budget.

The choice of battery has to be able to provide sufficient current to the motors to drive them effectively, for a sufficient period of time to complete the set tasks and compete. The power drawn by the other components will be negligible compared to the motors. The battery also has to be light enough so as not to use up too much of the sumobots weight budget – more Ah will mean heavier batteries, so a compromise will have to be made.

The sumobot will have to be able to sense its environment. It will have to be able to detect when it near the edge of the arena in order to take corrective action, and it will have to be able to detect obstacles/opponents. Two separate sets of sensors would suit these tasks as the sumobot will be detecting different conditions in each case. In the first case, the sumobot should be able to detect the white line in time to correct its course to prevent it from leaving the arena – this can most easily be achieved with IR detectors at the periphery of the sumobot. For the object/opponent detection, it would be advantageous for the sumobot to be able to detect both the direction and the range of the object. For successful direction detection, multiple sensors will most likely be needed. LIDAR or laser range detectors would be one of the more accurate technologies to use, but while the availability of cheaper detectors had increased recently, a lower price range LIDAR detector is still in the range of $80 (Dragon Innovation, 2014), so such sensors are not feasible on the budget allowed for this project. This leaves Ultrasonic sensors, IR sensors and bump sensors as the most viable options. Bump sensors would only be used in conjunction with other sensors, if at all, as the requirement of task 2 is evasion, and for task 1 and 3 it would be advantageous to detect the box/opponent before bumping into them. A choice will have to be made as to whether the sumobot required full or partial peripheral coverage with the sensors. At the very least 180° around the front of the sumobot would be desirable.

The brain of the bot is another consideration. Something low powered and lightweight would fit the design brief. It would be theoretically possible to have the sumobot run by a smartphone, tablet or a PC board, but these solutions would not be a practical use of the sumobots payload or power budget. The Raspberry Pi is a device recently on the market that offers a microcontroller with micro computing processing power, as the Raspberry Pi is a fairly new device, however, the online resources available to those working with it are not yet as diverse as for other available microprocessors. AVR microprocessors offer a platform that the student is already familiar with, while Arduino offer a range of easy to use microprocessors of different capabilities, and have the advantage of an active online community and freely available libraries specifically tailored for projects such as this, and a range of affordable plug in shields that can be used to drive motors and sensors, greatly simplifying the electronics involved in the project. If a motor shield is used, care has to be taken that the hardware is capable of handling the potential peak power demands of the motors being used. Likewise if a custom H-bridge or driver is used or designed, it will also have to be able to provide appropriate current without shorting.

The Sumobot needs to be able to process and react to its inputs effectively and in real time. It needs to operate in and switch between a number of modes. It needs to be able to effectively search the arena area – this may involve just spinning and scanning on the spot until an object is detected, or it may involve wandering and scanning repeatedly until an object is detected. On locating its obstacle/opponent, the sumobot has to make a fight/flight decision – for task one, the decision is obviously fight, and for two, flight, however for task three, if the opponent is detected approaching from the side, flight may be preferred, while if it is straight ahead, fight will probably be its default mode. On detecting the perimeter of the arena, the sumobot needs to be able to make effective correction to its course to avoid going over the edge – if it also detects its opponent or the box, this needs to also be factored in when planning the appropriate action to take. Additionally, if the sumobot has any design features beyond its basic requirements, how these features are activated or engaged will also have to be considered when building the sumobot and constructing its operating software. For example, if a flipper mechanism is used, when is it engaged and what triggers its activation? Or if an accelerometer is used to ensure that the sumobot is level, what evasive actions are taken if it starts to tip?

Finally all of this has to be achieved within the pre specified weight and budget constraints. The sum weight of all of the components cannot be 1g over a kilogram, but to have the best resistance to being pushed the design should also use its full weight allowance – the heaviest single items will most likely be the batteries and the gearmotors/servos, but the weight allowance will be the limiting factor on the size – and therefore Amp-hours – of the batteries used. The complexity of the design and the quality of the components available will be ultimately constrained by the $150 budget – which, for example, rules out the use of LIDAR sensors or certain high end motors. Any budget that can be saved in some areas can be invested in other components.

Updated Report 1

The final design retained much of the original proposed design, but also deviated in a number of key areas. These design changes mostly involved changes to the Sumobot's overall structural design, and the shedding of additional features and sensors. The changes were necessitated by the size and weight restrictions imposed by the project, as the overall design was modified to fit the components in to the required volume constraints, and features and additional sensors were dropped from the design.

Overall Design

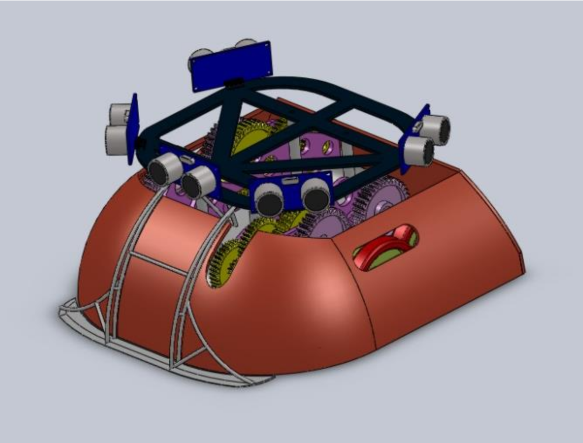


Figure 1: Initial Sumobot Design

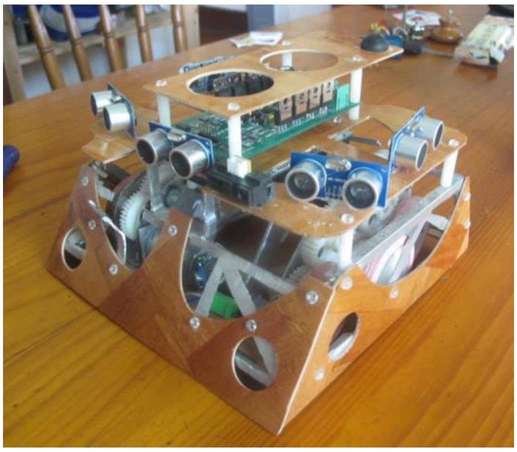


Figure 2: Final Sumobot Design

In the original design proposal, the structure of the Sumobot resembled a horseshoe crab, with a curved profile. A number of materials for the structure were being explored also, based on readily available recyclable materials. After considering stainless steel, melamine, HDPE, aluminium and laminated bamboo (from BBQ trays), an aluminium frame and gearbox chassis, with laminated bamboo cladding and chassis were settled on for the over-all design. Melamine was too brittle, and the stainless steel option was too weighty, the laminated bamboo had a better strength to weight ratio than all the other materials explored, but the aluminium was more versatile and easier to work than the bamboo for making the more complex structure, for the precision needed for the gear box, and lent itself to tapping bolting and screwing components to. The rounded edges on the original design were abandoned for square ones for ease of construction, and for more internal space for components. Plastic bolts were used for most of the fixings to reduce weight, but steel ones were used to attach the gear and motor mounts to the chassis where a more robust fixing was desired. All the structural elements were perforated as much as functionality would allow, in order to keep the over-all weight down.

Mechanical Design

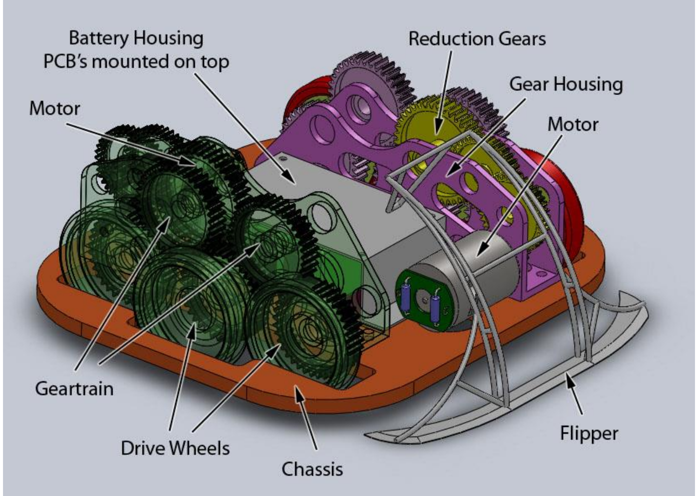


Figure 3: Illustration of proposed Sumobot Mechanical Details

Mechanically, the final design remained fairly true to the original proposed design, but the flipper mechanism had to be abandoned due to both size and weight constraints – to accommodate it, an extra 100g in weight was needed and an extra 15mm in length.

As discussed in the original project proposal, the driving motors are two RS 380SH RC hobby motors acquired from an RC Toys-R-Us jeep, with custom gearbox's and gear-trains assembled from gears also acquired from found RC vehicles. 3 wheels each were attached to the gear train, making the sumobot effectively six wheel drive for additional traction and stability. A differential drive will be used for steering, giving the sumobot good steering control – with the ability to effectively turn on the spot – but not so good for keeping in a straight line, which shouldn't affect its performance for the set tasks. For motor control, for simplicity of design a prefabricated H-Bridge was used – more detail in the next section.

The custom gearbox uses the following reduction gears: 2x(12T/38T) and 1x(12T/40T) – which equates to a gear ratio of ~33.4:1. An unloaded speed of 1ms^{-1} is desirable, which for a 50mm diameter wheel equates to: $\frac{1}{0.05 \times \pi} = 6.37 rps\ or\ 382 rpm$ .The gear ratio means that a motor of speed 12760 would be optimal. The motor that has been picked for this project claims 14,000 rpm at peak efficiency. Not allowing for inefficiencies, this would give this configuration a maximum speed of 1.1 ms^{-1} which is close enough to the desired maximum speed – inefficiencies will probably drop it closer to 1ms^{-1}, and, if need be, the maximum speed can be suppressed in the speed control code.

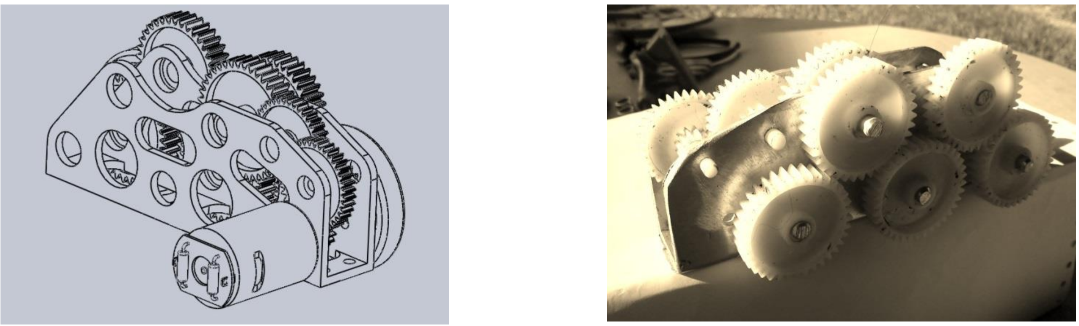


Figure 4: Motor, Reduction gear, geared drive train and housing assembly drawing, and (lower right) prototype assembly

The gear housing was machined out of 25x50x1.6mm aluminium RHS. The negative spaces in the housing also reduce the weight of that component by half. The axles for the gears were machined out of Delrin rod, as this has less than half the density of aluminium.

The wheels were machined out of Delrin, with custom silicon tyres for increased friction – currently test were conducted on a number of silicon mixes to see which can provide the most traction – this will help increase the potential acceleration of the sumobot as well as providing more resistance to opponents pushing force. Casting Silicon rubber was used for these tests, with some different additives such as corn starch and washing detergent were used, as recommended from online resources on making high traction wheels for sumobots (instructibles.com, 2014). The recipes sourced online seemed to be using tube silicon sealant from the hardware store in their mixes, rather than the 2 part casting silicon used here, so the end consistency of the finished product was likely quite different. From experimentation, the straight silicon mix, with a drop of detergent added had much better friction, though was more flexible, than the recipes using the corn starch additive. The diameter of the wheels is to be 50mm. For the purpose of our calculations here, and for convenience, we will consider the coefficient of static friction between the tyres and the arena to be 0.75 – which is analogous to the coefficient of static friction between vulcanised rubber and clean concrete (engineeringtoolbox.com 2014)

Assuming the sumobot uses its full weight allowance of 1kg, the force needed to overcome the static friction force is:

$$F_s = \mu_s F_N = \mu_s M g = 0.75 \,.\, 1 \,.\, 9.81 = 7.35N$$

As F = ma, the maximum acceleration we can ask from the Sumobot from stand still, without the wheels slipping, is 7.35/1 = 7.35ms^{-2} which would get the sumobot to its maximum speed of 1ms^{-1} in 0.135s. Also the maximum applied force desired at the point of wheel contact with the ground from each motor is 7.35/2 = 3.675N. The desired torque at centre of the wheel at start then is 3.675 x 2.5 = 9.2N.cm. A motor with greater torque than this is desirable to provide extra power when it comes to pushing opponents out of the ring, so some sort of additional calibratable electronic speed control may be necessary to ensure the motors always take at least 0.135s to ramp up to full speed from stationary. The minimum amount of torque needed on the motor to reach maximum possible acceleration then is 9.2/33.4 = 0.275N.cm or 28g.cm. The motor chosen for the project claims 85.2g.cm at peak efficiency and 700g.cm stall torque, so no shortage of power, and the sumobot will almost certainly need additional acceleration control in the speed control code for optimal performance.

On testing, the drive assembly performed well – there was no need for ramped acceleration in the code as there was no slippage on the wheels – the time it took for the motor to overcome internal frictions and inertia were enough to damp the acceleration sufficiently to avoid slip. The top speed of the sumobot seemed to be approximately 1ms^{-1}, and the bot managed to push 2kg of soymilk cartons across a smooth floor – though for this test standard alkaline batteries were being used, and the batteries drained fairly quickly with this load.

On testing, the sumobot travelled at a top speed of slightly above 1ms^{-1}, moving approximately 40cm in 0.3s, so travelling at around 1.2ms^{-1}.

Electrical and Motor Design

Originally the Arduino Uno was considered for the microprocessor to control the Sumobot, in the final design the Arduino Mega was chosen for the additional IO pins available.

Table 1: Arduino Mega 2560 specs (Arduino 2014)

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

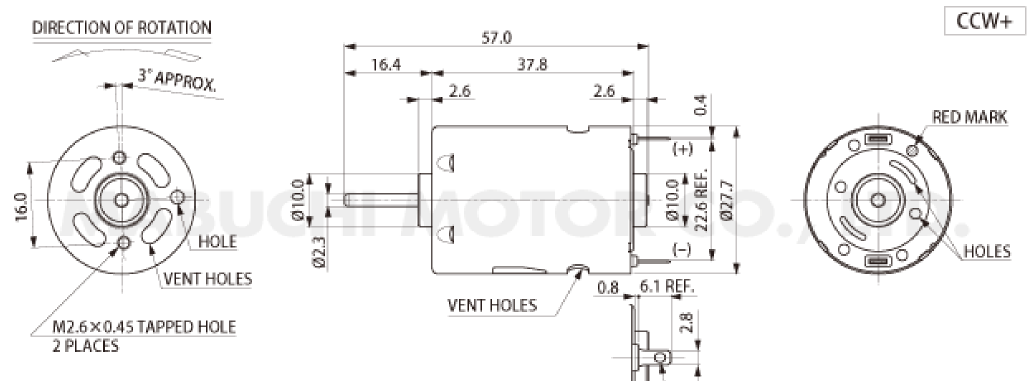
A number of motor options were considered, but in keeping with the obtainium criteria, found motors are preferred. Popolu gear motors were looked at, and a number of drive motors were extracted from printers and other devices for consideration. The printer motors were generally designed to run at higher voltages (30 – 48 VDC), and were generally heavier than desired (~200g)

A motor appropriated from a Toys-R-Us RC Jeep was settled on – a pair of spares was also found in another RC jeep acquired from council throw out, so it seems to be quite a common motor for larger RC toys. Further spares were ordered online in case of failure but the ones taken from the scavenged toy performed fine during testing.

The Motor is an RS 380 SH 3750:

Model No.	Main Performance									
	Rated Voltage	No Load		At Maximum Efficiency					At Stall	
		Speed	Current	Speed	Current	Torque	Output	Eff.	Current	Torque
	V	r/min	A	r/min	A	g.cm	W	%	A	g.cm
RS-380SH-3750R	9.0 CONST	16,500	0.45	13,966	1.96	85.2	12.67	71.68	14.67	700

Table 2: RS 380 SH Specifications (alibaba.com 2014)



取付けネジの長さはモータケース面より3.0以下
Usable machine screw length 3.0 max. from motor mounting surface.
安装螺丝的长度须从马达壳面算起3.0以内

Figure 5: RS 380SH Motor (alibaba.com 2014)

The larger version of this motor, the RS540 SH was considered, but at ~twice the weight (160g) and twice the current draw (~4A at maximum efficiency), it was considered too bulky, too power hungry and too heavy for this application.

As described in the previous section, reduction gears are used: 2x(12T/38T) and 1x(12T/40T) – which equates to a gear ratio of ~33.4:1, and which will reduce the

~14,000 rpm to ~420rpm, and increase the Torque to ~2.85kg.cm at maximum efficiency and ~25kg.cm at stall torque.

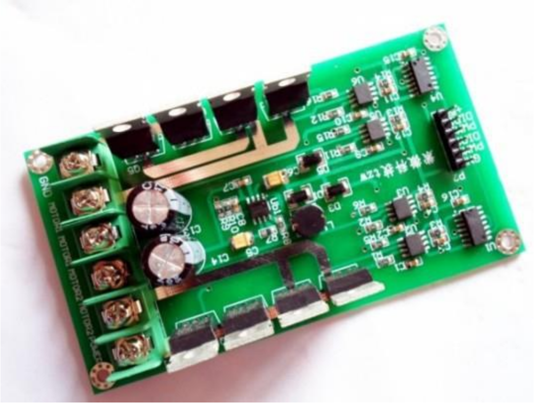


Figure 5: H-bridge DC MOSFET IRF3205 30A (ebay.com.au 2014)

A number of prefabricated H Bridge boards were considered, but in the end, considering the potential power draw of the motors, the board decided on was a Dual Motor Driver Module board H-bridge DC MOSFET IRF3205 30A (ebay.com.au 2014). The stated specification for this board were; Rated voltage: 3v-36v, Rated Current: 10A and Peak current: 30A (ebay.com.au 2014) which is ample for the motors used in this design.

A 2 position switch was used for setting the sumobot mode – 1 and 2 for tasks one and two, and 0 for competition mode. A momentary push button was used for start and reset, and a 3 colour LED was chosen for status display. Originally, a sound chip was intended to be assimilated into the design in order to play “Flight of the Valkyries” in attack mode, the “Benny Hill” theme for evade mode, and “Eye of the Tiger” for competition mode. Unfortunately this idea was dropped due to weight considerations and coding complexity and schedule.

So, given these various components, as well as the status LED’s and sensors, if the motors are running at peak efficiency, rounding up to assume they are under some load, the Sumobots normal running current will be around 5A. The first two tasks will take four minutes, and with around ten other students competing in the tournament, if the sumobot performs well, it may have to provide 1 hour of continuous running time. To achieve this without a battery change would require 5Ah of capacity – this may be a little optimistic given the weight allowance, but half that should be achievable.

12V 4.8AH LiPo batteries were considered for the power supply for the Sumobot (ebay.com.au 2014), unfortunately, when these were ordered, one of them didn’t work at all, putting out 3V instead of 12V, and the other malfunctioned while testing. Also at 160g, these batteries would most likely have pushed the Sumobot over the weight restriction. Prior to demonstration day there was no time left to obtain a more custom RC battery, so three 9V Energizer LiPo batteries in parallel were chosen. Each of these batteries is reported to have ~700mAh capacity, so four in series should provide ~2.1Ah. These batteries are non-rechargeable, and one spare set would be required. Each one of these batteries weighs 34g and costs around $15, so the bank of 3 used added up to ~100g payload and $45 of the budget (energizer.com 2014).

While the Arduino board claims to be able to operate with a 12V power supply, running it with such a source can lead to overheating and damaging of components (Adrian Carter 2012). To avoid this a 12V to 5V 3Amp converter power supply was used to power the Arduino and provide power for all the lower voltage components, such as a the sensors and the data side of the H bridge (ebay.com.au 2014).

The motors decided on will provide about 10kg of force, per wheel, at the circumference of the wheel at peak torque – which if we apply F=Ma, gives us approximately 20ms-2 acceleration, or enough to reach maximum velocity of 1ms-1 in about 1/12 of a second – in all likely hood this could be too much for the traction of the wheels, so some ramping may have to be incorporated in the software. Provided the sumobot performs optimally, and all the components are as good as it says on the packet, the Sumobot, if not under any external load, should travel 50m in 50s. If we continue with our estimate of 5A of current needed to run the sumobot optimally, then that 50m run would use 69.4mAh of battery capacity – if the 2.1 Ah battery packs are being used, and they are at full capacity and performing as advertised, they will have 4.0306Ah left.

Sensor Design

For the primary sensors for the obstacle/opponent sensors an array of 5 HC-SR04 ultrasonic sensors mounted above the other components was chosen. These are economic, commonly available ultrasonic sensors, with the added advantage of having a small box full of them sitting in the workshop, so also fulfil the “obtainium” criteria.

Figure 6: HC-SR04 Ultrasonic Sensor *(Cytron Technologies, 2013)*

HC-SR04 Specifications *(Cytron Technologies, 2013)*:

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" - 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm

The proposed layout for the ultrasonic array provides good coverage for the front of the sumobot with a blind spot to the rear and a potential blind spot to either side. Given that it is expected that the Sumobot will be in constant motion, the lateral blind spots are not expected to be an issue. Additional IR sensors are being considered in anticipation of rival sumobots using some sort of physical damping in their design to confuse the ultrasonic sensors. The effective range for the chosen sensors is up to 4000mm – it can therefore detect objects outside of the arena, so a limit to the range which it reacts to will have to be included in the code. Initially a sharp IR range finder was to be used to supplement the US range finders, in case any of the other sumobots were to attempt US blocking. In its final configuration on demonstration day, the IR sensor and the two lateral/rear facing US sensors were dropped from the design, as the code to allow for them had not been written and tested in time.

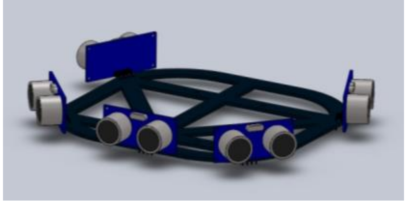


Figure 8: Ultrasonic Array

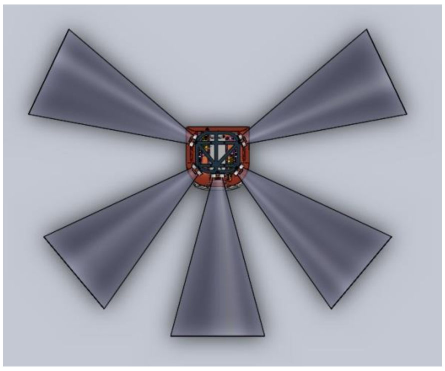


Figure 7: Ultrasonic Coverage (to 500mm range)

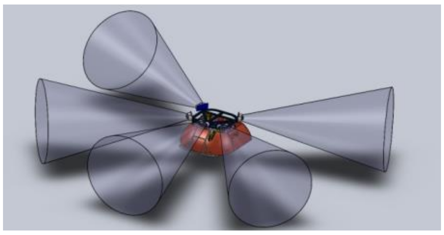
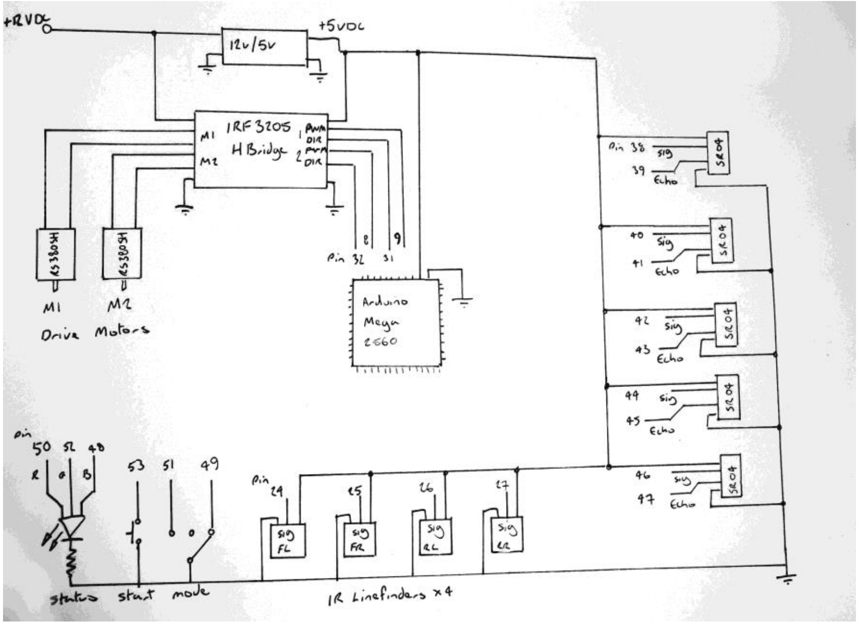


Figure 9: Ultrasonic Coverage

The IR sensors originally considered for line detection (IR Sensor YL-63 (banggood.com 2014)) were more suitable for range finding, so the Grove line finder from Seeed Studio was settled on instead – this unit is more suited to the desired application and also a more compact unit, so much more suited to this purpose. It also lends itself to simpler programming, as it just returns a HIGH (dark surface) or a LOW (light surface) with a variable resistor for adjusting the range between 1.5cm and 5cm (seeedstudio.com)

Schematic



Programming Strategy

The steering strategies for the sumobot included;

"brake()" command, which would instantly set both motor speeds to zero

"Run()" command – commanding the sumobot to move straight forward or reverse at a set speed – its inputs were velocity (0 – 255 for the PWM) and direction (Forward or Reverse)

"spin()" command – commanding the motors to run at the same speed but in opposite directions – it's inputs were velocity (0 – 255 for the PWM) and direction (Left or Right for direction of spin.

And "turn()" command – commanding the motors to run at differential speeds, but in the same direction - it would determine the turn radius with two inputs turn speed numerator and turn speed denominator – so if a TSpeedN of 2 and a TSpeedD of 5 was entered, the inside motor would run at 2/5 the speed of the outside motor. The inputs for this command then were velocity (0 – 255 for the PWM), TSpeedN, TSpeedD, turn direction (left or right) and direction (forward or backward).

These commands would set the motors running in the desired mode continuously, until alternative commands were issued. In addition to these there were "runTime()", "spinTime()" and "turnTime()" commands, which would have an additional "time (ms)" input where the steering strategy would be executed for the predetermined time then stop. This was implemented using clocking function, using the millis() command, instead of a delay so as not to use up processing time.

To control the Ultrasonic Sensors, the library newPing was implemented (Tim Eckel, 2012). newPing uses a timing function instead of delays also, and an interrupt function for processing inputs, so allows for faster polling of multiple US sensors without freezing or slowing other processes.

In order to operate the line detector IR sensors the PinChangInt library (Arduino 2014) was used, which allows for state changes on multiple pins to be applied to a single interrupt, and vice versa. All four sensors were applied to an interrupt, then each sensor polled and a suitable evasion steering strategy implemented.

For the first task, the sumobot was programmed to spin and search for the box, then, once located, the box was to be pushed straight out of the ring, with the sumobot stopping at the edge of the arena. For task two, the sumobot was to turn 180°, then reverse to the edge of the ring, poling its US until the box was detected, at which point it would take evasive measures, and reverse to another point on the edge of the ring, waiting for the box to appear again. For the competition strategy, the intention was for the sumobot to locate its opponent, and then steer round the side of it, turning and attacking when it detected its opponent in the lateral/rear facing sensors. The purpose of this strategy was to both avoid getting scooped by opponents ramps, and to avoid having to work against the opponents motors. Due to time constraints and a desire to keep the code simple, the strategy opted for in the end was more similar to task 1, where the sumobot would, search and attack, then repeat the process in case the initial attack was unsuccessful.

Robot Performance

On the demonstration day, the Sumobot failed to perform at all, due to code failure. A number of factors contributed to this failure, but at their root is not managing to allow enough time at the end of the schedule for comprehensive testing and debugging of the code. Modules of the code were tested during assembly, and were made to function satisfactorily. The various steering functions ran well during testing, and the Sumobot demonstrated impressive power, speed and agility during the motor control testing phase.

The problems really arose when, three days prior to demonstration and final assembly and compiling of the code, during some motor control tests, the motor controller short circuited, the spares having been left in Sydney, the final stages of assembling the program were conducted effectively blind, as I was unable to test the operation of the robot in real world situations as the code was compiled.

In an attempt to overcome this obstacle, the outputs were observed using a logic probe, a DMM, a nano digital oscilloscope, and outputting to a laptop through serial communication. Again, the sections of code seemed to be operating as expected during testing, but some errors occurred during compiling and final assembly of both code and bot, as additional features were added to the code. At the time of writing an ongoing debugging process is being conducted, and a demonstration of a functional sumobot is to follow.

Each stage of the robot development took longer than anticipated or planned for, which clearly left not enough time for effective coding, debugging and trouble shooting in the final stages of development

Suggested Improvements

At the time of writing, the code has yet to be fully debugged and tested, so it is hard at this point to suggest many areas for improvement. As described earlier in this report, much of the additional features intended in the original sumobot design were abandoned due to size and weight constraints. The flipper mechanism was dropped due to both of these, as it required an additional 100g in weight, and 15mm in length to be compatible.

Additional sensors (rear and side pointing ultrasonics, and an additional IR range sensor as back up to overcome any US blocking techniques that may have been used by other students, such as the use of sound absorbing cladding) were dropped, in part to reduce weight, but also to reduce programming complexity.

Much of the issues with volume and weight arose due to the bulkiness and weight of the custom built gearboxes and gear trains, these parameters were in turn determined by the dimensions of the components that were had to hand. In order to overcome some of the subsequent challenges presented by this design choice – including the development time used up by this labour intensive procedure and the resulting loss of code development time, several other approaches could have been favoured. The most obvious would be the use of off the shelf components, such as a more compact, contained gear box, or a whole gear-motor assembly. The motors used were cost effective and powerful, and most of the gear-motors that were looked at during the design phase were less powered, but there may have been more compatibly gearbox units available. Instead of a 5 cog gear train to power the 6WD wheels, a belt and pulley system, or an off the shelf track assembly may have been more compact and practical, and simpler to assemble. These modification may have afforded the same traction and power while driving, but allowed enough extra space and weight for additional features such as the flipper mechanism.

The US sensors were mounted rather high off the ground, and may have caused blind spots given the average height of the other sumobots. It would be better to incorporate the US mounts lower down on the sumobots body. Again, this may have not been possible with the current component configuration given the space taken up by the gearbox/drive train assembly.

The custom tyres had a reasonably amount of traction, but were a little stretchy and would work their way off the wheels too easily - a better attachment for the tyres to the wheels would be desirable, and further experimentation with optimal silicon mixes for the wheels may also yield better tyres.

A custom design PCB to house the microprocessor would also help to reduce the volume and weight taken up by unnecessary components and board space.

Further research into the optimal power supply would also help the overall design – the batteries originally chosen for the project proved to be very unreliable – a problem in the quality of the supplier rather than the choice of battery itself – and were also a bit heavier that desired, and in all likelihood, would have put the finished bot a bit over its allowed weight, even with the extra components and features stripped. The original batteries considered for the project reportedly had 4.8AH of capacity – where 1.5 to 2AH would have sufficed for this project.

The steering strategies developed in the code were written with more complex navigation and steering strategies in mind, such as ramping the acceleration in the motors and possibly applying PID control for smoother action, as well as expanding on the search routines. At the time of writing, the code is still not functioning properly, so it is hard to propose further improvements to the code structure at this point.

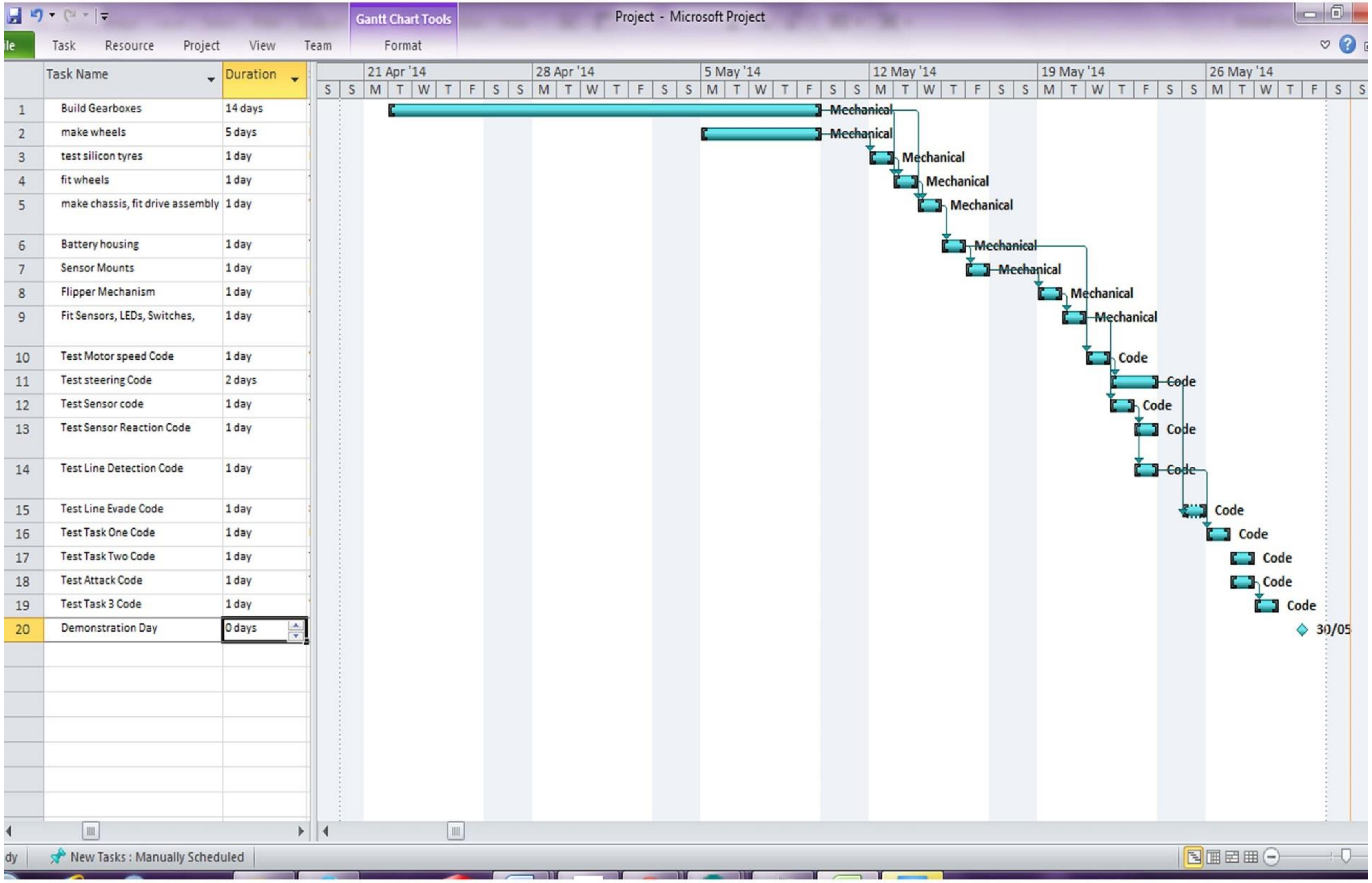
Reflection on Area of Expertise

The Arduino programming environment was a new one to this writer, and seemed to be the one being used by the majority of the students. As the programming was the final stage for most students projects, I wasn't called upon for advice in this area much, as I presume most of the students would be fairly intensely involved in getting their projects working at that stage, and more likely to be seeking advice to those more immediately at hand, in the case of the on campus students. I found the Arduino environment much more accessible and easier to use than previous microprocessor experiences I had had, and the availability of additional libraries was also a major advantage. I feel my expertise in this area improved as the unit progressed, and is improving still, and would have liked to have been more of a helpful resource to my fellow students.

Bill Of Materials

Item	Quantity	Supplier	Material	Cost
RS 380SH Motors	2	Obtainium	Electrical	$ -
Gears	multiple	Obtainium	Delrin	$ -
Gearhousing	2	Obtainium	Aluminium	$ -
Body Frame	1	Obtainium	Aluminium	$ -
Chassis	1	Obtainium	Laminated Bamboo	$ -
Cladding	1	Obtainium	Laminated Bamboo	$ -
Sensor Mount	1	Obtainium	Laminated Bamboo	$ -
Motor Controller	1	Ebay	Electrical	$ 22.00
12V/5V Converter	1	Ebay	Electrical	$ 9.00
Ultrasonic Sensors	5	Stock	Electrical	$ -
IR Line Finders	4	Seeed Studios	Electrical	$ 15.60
Batteries	3	Office Works	9V Enegizer LiPo	$ 45.00
Assorted Electrical	multiple	Jaycar	Electrical	$ 20.00
Assorted Fixings	multiple	Jaycar	Plastic	$ 12.00
Arduino Mega	1	Stock	Electrical	$ -
Total Cost				$ 123.60

Revised Workflow



	Task Name	Duration
1	Build Gearboxes	14 days
2	make wheels	5 days
3	test silicon tyres	1 day
4	fit wheels	1 day
5	make chassis, fit drive assembly	1 day
6	Battery housing	1 day
7	Sensor Mounts	1 day
8	Flipper Mechanism	1 day
9	Fit Sensors, LEDs, Switches,	1 day
10	Test Motor speed Code	1 day
11	Test steering Code	2 days
12	Test Sensor code	1 day
13	Test Sensor Reaction Code	1 day
14	Test Line Detection Code	1 day
15	Test Line Evade Code	1 day
16	Test Task One Code	1 day
17	Test Task Two Code	1 day
18	Test Attack Code	1 day
19	Test Task 3 Code	1 day
20	Demonstration Day	0 days

References

alibaba.com 2014 http://www.alibaba.com/product-detail/9v-dc-motor-RS-380SH-3750R_273187656.html viewed 1st June 2014

Arduino 2014 http://arduino.cc/en/Main/arduinoBoardMega2560 viewed 1st June 2014

Arduino 2014 http://playground.arduino.cc/Main/PinChangeInt viewed 1st June 2014

Carter, Adrian 2014 "Fuelling Your Arduino - Why you should use an External Power Breakout", http://www.australianrobotics.com.au/blogs/adrian-carter viewed 1st June 2014

Banggood.com 2014 http://www.banggood.com/Infrared-Obstacle-Avoidance-Sensor-For-Arduino-Smart-Car-Robot-p-72965.html April 2014Cytron Technologies 2014 HC-SR04 User's Manual 2013

Deakin University 2014 " SEM433 Mechatronic Design Project Details" Deakin University 2014

Dragon Innovation 2014 http://www.dragoninnovation.com/projects/32-lidar-lite-by-pulsedlight Viewed 19th April 2014

Ebay.com.au 2014 http://www.ebay.com.au/itm/261442251377?ssPageName=STRK:MEWNX:IT&_trksid=p3984.m1439.l2649 viewed 1st June 2014

Ebay.com.au http://www.ebay.com.au/itm/350585953725?ssPageName=STRK:MEWNX:IT&_trksid=p3984.m1439.l2649 viewed 1st June 2014

ebay.com.au 2014 http://www.ebay.com.au/itm/331032292713?ssPageName=STRK:MEWNX:IT&_trksid=p3984.m1439.l2649 viewed 1st June 2014

Eckel, Tim 2012 https://code.google.com/p/arduino-new-ping/ viewed 1st June 2014

Energizer.com 2014 http://data.energizer.com/PDFs/la522.pdf viewed 1st June 2014

engineeringtoolbox.com 2014 http://www.engineeringtoolbox.com/friction-coefficients-d_778.html Viewed 19th April 2014 instructibles.com 2014 http://www.instructables.com/id/Grippy-robot-wheels/ Viewed 1st June 2014

seeedstudio.com 2014 http://www.seeedstudio.com/wiki/index.php?title=Twig_-_Line_Finder Viewed 1st June 2014

Sumobot Code

This is a copy of the code at the time of demonstration day – note that this code was not functional, and therefore incomplete. A copy of the functional code will be submitted with the video of the functional sumobot, to follow later.

```
//Dillon MacEwan student ID 211285035
//sumobotcode
//written for SEM433 Mechatronic Design, Deakin University 2014
//Newping library and code written by Tim Eckel code.google.com/p/arduino-new-ping/
//Pin Change interrupt library source: code.google.com/p/arduino-pinchangeint/
#include <NewPing.h>
#include <PinChangeInt.h>



#define SONAR_NUM     3 // Number or sensors.
#define MAX_DISTANCE 100 // Maximum distance (in cm) to ping.
#define PING_INTERVAL 33 // Milliseconds between sensor pings (29ms is about the min to avoid cross-sensor echo).

unsigned long pingTimer[SONAR_NUM]; // Holds the times when the next ping should happen for each sensor.
unsigned int cm[SONAR_NUM];         // Where the ping distances are stored.
uint8_t currentSensor = 0;          // Keeps track of which sensor is active.

NewPing sonar[SONAR_NUM] = {     // Sensor object array.
  NewPing(38, 39, MAX_DISTANCE),
  NewPing(40, 41, MAX_DISTANCE),
  NewPing(42, 43, MAX_DISTANCE), // Each sensor's trigger pin, echo pin, and max distance to ping.
//  NewPing(44, 45, MAX_DISTANCE),
//  NewPing(46, 47 , MAX_DISTANCE),
};

//set pushbutton pins
const int Start = 53;
const int Mode1 = 49;
const int Mode2 = 51;
//set LED pins
const int Red = 50;
const int Blue = 48;
const int Yellow = 52;
//set motor pins
const int Dir1 = 30;
const int Dir2 = 31;
const int M1Speed = 8;
const int M2Speed = 9;
//set line detector pins
const int FrontL = 24;
const int FrontR = 25;
const int RearL = 26;
const int RearR = 27;
//Set Ultrasonic Pins
//const int FrontTrig = 38;
//const int FrontEcho = 39;
//const int FLeftTrig = 40;
//const int FLeftEcho = 41;
//const int SLeftTrig = 42;
//const int SLeftEcho = 43;
//const int FRightTrig = 44;
```

```
//const int FRightEcho = 45;
//const int SRightTrig = 46;
//const int SRightEcho = 47;

const int Forward = HIGH;
const int Back = LOW;
const int Left = HIGH;
const int Right = LOW;
const int LeftUS = 0;
const int CentreUS = 1;
const int RightUS = 2;

long taskTime = 1000;
int search = 0;
long starting = millis();



int mode = 0;
int start =  LOW;



void setup()
{
  //initialise all pins
  //initialise LED pins
  pinMode(Red, OUTPUT);
  pinMode(Blue, OUTPUT);
  pinMode(Yellow, OUTPUT);
  //intitialise button pins
  pinMode(Start, INPUT);
  pinMode(Mode1, INPUT);
  pinMode(Mode2, INPUT);
  //initialise motor pins
  pinMode(Dir1, OUTPUT);
  pinMode(Dir2, OUTPUT);
  pinMode(M1Speed, OUTPUT);
  pinMode(M2Speed, OUTPUT);
  //initialise line sensor pins
  pinMode(FrontL, INPUT);
  pinMode(FrontR, INPUT);
  pinMode(RearL, INPUT);
  pinMode(RearR, INPUT);

  digitalWrite (Yellow, LOW);
  digitalWrite (Red, LOW);
  digitalWrite (Blue, LOW);

 search = 0;



  if (digitalRead(Mode1) == HIGH)
    mode = 1;
  else if (digitalRead(Mode2) == HIGH)
    mode = 2;
  else
    mode = 0;
    attachInterrupt(FrontL, &edgeFind, FALLING);
    attachInterrupt(FrontR, &edgeFind, FALLING);
    attachInterrupt(RearL, &edgeFind, FALLING);
    attachInterrupt(RearR, &edgeFind, FALLING);

  pingTimer[0] = millis() + 75;           // First ping starts at 75ms, gives time for the Arduino to chill before starting.
  for (uint8_t i = 1; i < SONAR_NUM; i++) // Set the starting time for each sensor.
    pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;

    while (start == LOW)//wait for start button to be pushed
  {
    int pushStart = digitalRead (Start);
    if (pushStart == 1)
    start = HIGH;
    delay(50);

  }

  long starting = (millis() + 5000);

  delay (5000);

}

void loop()
{



  if (mode == 1)
    mode1();
    else if (mode == 2)
      mode2();
    else if (mode == 0)
      mode0();

int pushStart = digitalRead (Start);
    if (pushStart == 1)
    {
    delay(50);
    softReset();
    }
  if ((millis() - starting) > taskTime)
    softReset;


}

//mode functions
void mode0() //compete mode
  {
    taskTime = 200000;
    digitalWrite (Blue, HIGH);
    runTime (255, HIGH, 300);
    spinTime (100,Right,300);
    for (uint8_t i = 0; i < SONAR_NUM; i++)
    { // Loop through all the sensors.
    if (millis() >= pingTimer[i])
    {         // Is it this sensor's time to ping?
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will be pinged.
      sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before starting a new ping (insurance).
      currentSensor = i;                          // Sensor being accessed.
      cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo for this sensor.
      sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to look
for echo).
    }
    }
     search++;
    if (search == 2)
      softReset;

  }

void mode1() //attack mode
  {
    taskTime = 150000;
    digitalWrite (Red, HIGH);
    runTime (255, Forward, 300);
    spinTime (100,Right,300);
    for (uint8_t i = 0; i < SONAR_NUM; i++)
    { // Loop through all the sensors.
    if (millis() >= pingTimer[i])
    {         // Is it this sensor's time to ping?
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will be pinged.

      sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before starting a new ping (insurance).
      currentSensor = i;                          // Sensor being accessed.
      cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo for this sensor.
      sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to look
for echo).
    }
    }

  }

void mode2() //evade mode
  {
    taskTime = 70000;
    digitalWrite (Yellow, HIGH);
    runTime (255, Forward, 300);
    spinTime (255,Right,50);
    while (digitalRead(RearL)==HIGH && digitalRead(RearR)==HIGH)
    {
      run (255, Back);
    }
    brake();
    for (uint8_t i = 0; i < SONAR_NUM; i++)
    { // Loop through all the sensors.
    if (millis() >= pingTimer[i])
    {         // Is it this sensor's time to ping?
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will be pinged.

      sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before starting a new ping (insurance).
      currentSensor = i;                          // Sensor being accessed.
      cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo for this sensor.
      sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to look
for echo).
    }
    }
  }

//motor control functions

void brake() //Stops both motors
{
analogWrite(M1Speed, 0);
analogWrite(M2Speed, 0);
}

void spin (int Vel, int Dir) //spins on the spot
{
  int OpDir;
  if (Dir == HIGH)
    OpDir = LOW;
    else
    OpDir = HIGH;
  analogWrite(M1Speed, Vel);
  analogWrite(M2Speed, Vel);
  digitalWrite(Dir1, Dir);
  digitalWrite(Dir2, OpDir);
}

void run (int Vel, int Dir) //runs in a straight line, forward or reverse
{
  analogWrite(M1Speed, Vel);
  analogWrite(M2Speed, Vel);
  digitalWrite(Dir1, Dir);
  digitalWrite(Dir2, Dir);
}

//steers left or right Tdir 0 = left 1 = right
void turn (int Vel, int TSpeed, int Tdir, int Dir)

{
  int TSpeed1;
  int TSpeed2;
  if (Tdir = 0)
    {
      TSpeed1 = TSpeed;
      TSpeed2 = 1;
    }
    else
      {
        TSpeed1 = 1;
        TSpeed2 = TSpeed;
      }
  analogWrite(M1Speed, Vel/TSpeed1);
  analogWrite(M2Speed, Vel/TSpeed2);
  digitalWrite(Dir1, Dir);
  digitalWrite(Dir2, Dir);
}

void spinTime (int Vel, int Dir, long Time) //spins on the spot
{
  int OpDir;
  if (Dir == HIGH)
    OpDir = LOW;
    else
```

```
   OpDir = HIGH;
 long startTime = millis();
 long currentTime = millis();
 while (currentTime - startTime < Time)
  {
   analogWrite(M1Speed, Vel);
   analogWrite(M2Speed, Vel);
   digitalWrite(Dir1, Dir);
   digitalWrite(Dir2, OpDir);
   currentTime = millis();
  }
  brake();
}

void runTime (int Vel, int Dir, long Time) //runs in a straight line, forward or reverse
{
 long startTime = millis();
 long currentTime = millis();
 while (currentTime - startTime < Time)
  {
   analogWrite(M1Speed, Vel);
   analogWrite(M2Speed, Vel);
   digitalWrite(Dir1, Dir);
   digitalWrite(Dir2, Dir);
   currentTime = millis();
  }
  brake();
}

//steers left or right Tdir 0 = left 1 = right
void turnTime (int Vel, int TSpeedN, int TSpeedD, int Tdir, int Dir, long Time)

{
 int TSpeed1N;
 int TSpeed2N;
 int TSpeed1D;
 int TSpeed2D;
 if (Tdir = 0)
   {
     TSpeed1N = TSpeedN;
     TSpeed2N = 1;
     TSpeed1D = TSpeedD;
     TSpeed2D = 1;
   }
   else
     {
       TSpeed1N = 1;
       TSpeed2N = TSpeedN;
       TSpeed1D = 1;
       TSpeed2D = TSpeedD;
     }
 long startTime = millis();
 long currentTime = millis();
 while (currentTime - startTime < Time)
  {
     analogWrite(M1Speed, (Vel/TSpeed1D)*TSpeed1N);
     analogWrite(M2Speed, (Vel/TSpeed2D)*TSpeed2N);
     digitalWrite(Dir1, Dir);
     digitalWrite(Dir2, Dir);
     currentTime = millis();
  }
  brake();
}

void echoCheck()

{
 if (sonar[currentSensor].check_timer())
 {
   if (mode == 0 || mode ==1)
   {
     if (currentSensor == CentreUS)
     {
       while (digitalRead(FrontL)==HIGH && digitalRead(FrontR)==HIGH)
       {
       run (255,Forward);
       }
       brake();
       turnTime (255,1,2,Right,Back,200);
       turnTime (255,1,2,Left,Back,200);
       spinTime (100,Right,300);
   for (uint8_t i = 0; i < SONAR_NUM; i++)
   { // Loop through all the sensors.
   if (millis() >= pingTimer[i])
   {        // Is it this sensor's time to ping?
     pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will be pinged.

     sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before starting a new ping (insurance).
     currentSensor = i;                          // Sensor being accessed.
     cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo for this sensor.
     sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to look
for echo).
     search++;
   }
     }
   }
   if (mode == 2)
   {
     if (currentSensor == CentreUS || currentSensor == LeftUS)
     {
       turnTime (255,1,2,Right,Forward,200);
       turnTime (255,1,2,Left,Forward,200);
     }
     else if (currentSensor == RightUS)
     {
       turnTime (255,1,2,Left,Forward,200);
       turnTime (255,1,2,Right,Forward,200);
     }
   }
 }
 }
}


void edgeFind()
{



 {
   if (mode != 0)
  {
   if (FrontL == LOW)
    {
     turnTime (255,1,2,Right,Back,200);
     turnTime (255,1,2,Left,Back,200);
    }
   else if (FrontR == LOW)
    {
     turnTime (255,1,2,Left,Back,200);
     turnTime (255,1,2,Right,Back,200);
    }
  }
  if (mode != 2)
  {
   if (RearL == LOW)
    {
     turnTime (255,1,2,Right,Forward,200);
     turnTime (255,1,2,Left,Forward,200);
    }
   else if (RearR == LOW)
    {
     turnTime (255,1,2,Left,Forward,200);
     turnTime (255,1,2,Right,Forward,200);
    }
  }
 }
}

void softReset()
{
 delay(100);
 asm volatile ("  jmp 0");
}
```