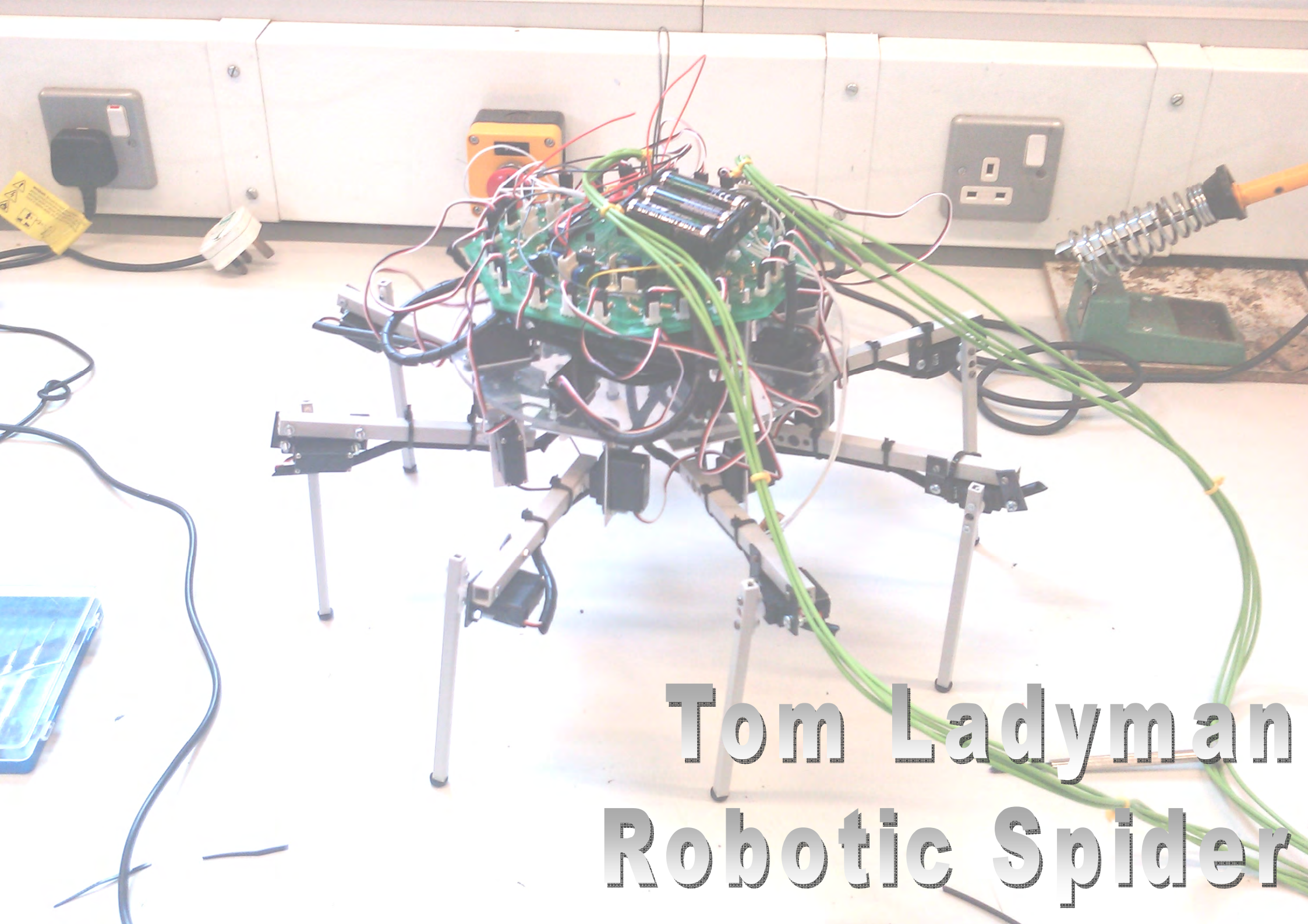

Tom Ladyman
Robotic Spider

Contents

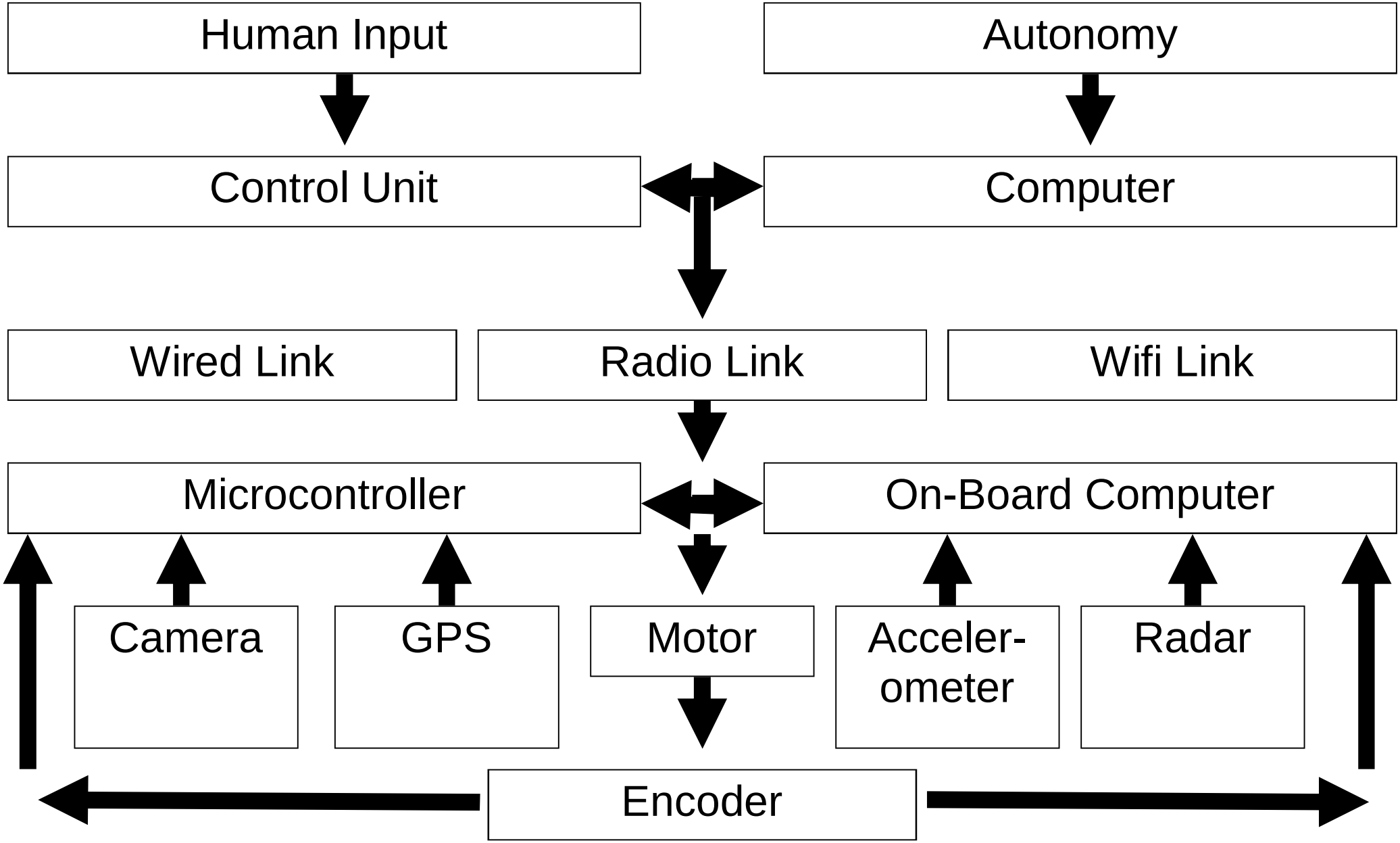
Section 1 - Planning - Initial Design Brief and Specification

Design Brief

Disaster zones are often left extremely dangerous, even long after the initial devastation has occurred. A nuclear bomb site will be left uninhabited for decades and a town levelled by a hurricane could stay as rubble indefinitely. To safely survey the area, an all terrain vehicle is required. The military and intelligence gathering services also need small, light and discrete vehicles to be used for the gathering of intelligence in politically awkward situations. I want to design and make a small, electronically powerful, all terrain vehicle to be used for both of these purposes. The cost should be relatively low, as both circumstances contain a risk factor. This is only relative, however, to hugely expensive alternatives.

Initial Specification

- The robot should be remotely operated.
- It should be capable of sending back information, including a camera feed, data about the surroundings and navigational aids.
- It should be able to cover all terrain, including rough ground such as forests and severe inclines.
- Steps may need to be overcome, so there should be a mechanism to allow this.
- It should have enough power such that it is never left stranded.
- It should be light so that it can't cause any damage to an already devastated area and it can move subtly.
- It must be fast and agile; able to manoeuvre in enclosed spaces.
- There should be a way of manufacturing it in an industrial environment.
- It must have reliable operation.
- The controls must be simple to use but also versatile and powerful.
- The wireless link between device and controller must be reliable.
- It should be made from parts which are easily sourced, in case of damage.
- Water resistivity would be incredibly useful.
- If possible, a microphone output should be also send back.
- If it runs out of power, it must have a backup system to stop it damaging itself or surroundings.
- Reliable and powerful navigational aids should be on board, otherwise it is no use for surveying.
- To assist with the surveying, it must carry equipment to monitor the environment.
- It must be fairly hardy and able to withstand minor damage
- The ground it moves along might well be wet or slippery, and it must overcome this.
- It must not be possible to disorient the moving mechanisms, ie, falling over, rolling etc.


Human Input
Autonomy
Control Unit
Computer
Wired Link
Radio Link
Wifi Link
Microcontroller
On-Board Computer
Camera
GPS
Motor
Acceler-
ometer
Radar
Encoder

Section 1 - Planning - Time Plan

Job/Month	September	October	November	December	January	February
Planning						
Mechanical Research						
Electronic Research						
Prototyping						
Mechanical Construction						
Electronic Construction						
Programming						
Fault Finding						
Casing						
Controller						
Testing						
Written Work						

The quad bike is the most commonplace of all of the all terrain vehicles I have researched. It is used mainly for off road sport and thrill seeking, rather than a mode of transport. It is suitable for all terrain coverage due to its high ground clearance, large and rugged wheels and specialised suspension.

The handlebars allow for a lot of manoeuvrability and make it intuitive and easy to use. Strong brakes are also an important feature to note, as some models can reach high speeds.

A wide wheel base make the quad bike very stable, which is very useful for all terrain driving, particularly for encountering slopes and ditches.

Wheels however, mean that in incredibly slippery areas, the quad bike can be very difficult to control and steps are an obstacle to be avoided.

Tanks however, use tracks instead of wheels. These cannot be punctured by a bullet, or rough terrain. They also allow the tank to take on steps, however, this is also due to the scale of them.

The other advantage of tracks is that it allowed the vehicle to turn on the spot, allowing complete rotation even in a confined space. The turret also caught my attention, making it as useful as a multidirectional vehicle, as it can swing around.

The final all terrain vehicle I looked at is a robot from Star Wars, known as a walker or AT AT. Although these are fictional, they provide a good basis for analysis. Having four long legs allow the robot to walk over any obstacle, including steps of a similar size to the robot itself. Such long legs, however, can be a disadvantage, as it looks very unstable, particularly from a side on attack.

Another point of note is how few axis of movement there are on each leg. This allows for very little other than a straight walking motion. I can also see no mechanism for turning.

I like the idea of legs very much, because literally nothing stands in their way, but I like aspects of the other vehicles very much too, in particular, the multi directional ability of the tank and it's degree of manoeuvrability, as well as the stability of the quad bike. I will research some vehicles with legs and combine these features.

Section 1 - Planning - Similar Products - Legged Vehicles

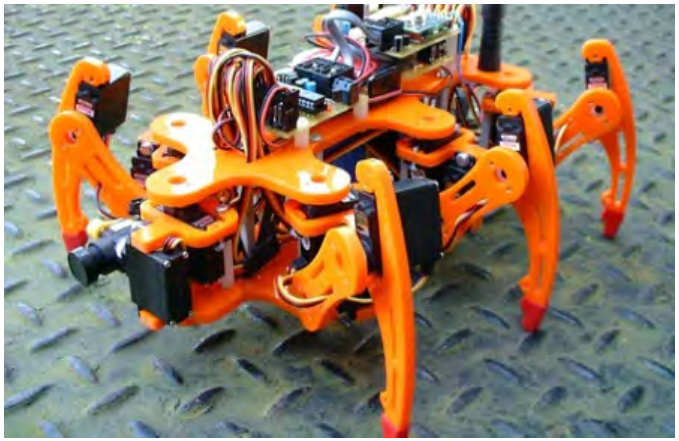
The photo on the left is of a quadrapod; a four legged robot. The one I chose to look at has very little in the way of electronics or features, this allows me to focus on the mechanisms. The base of each leg pivots along the longest dimension of the main chassis. This base servo is attached directly to another servo which controls the slant of the leg. A third and final servo controls the bend in the leg. This is a very interesting concept, as it emulates a ball and socket. I definitely think there is a more efficient design, however.

Asimo is one of the most famous robots in the world. Developed by Honda, the humanoid is capable of all sorts of advanced interactions with humans, including speech, learning name of objects and facial recognition.

His legs however, are the focus of my research. The start with a true ball and socket at the top and have a hinge joint halfway down. This is not a standard hinge though, it also slants, so it is effectively an elliptical sphere and socket joint. The ankle is another ball and socket and controls the whole foot. His walk is incredibly accurate and stable.

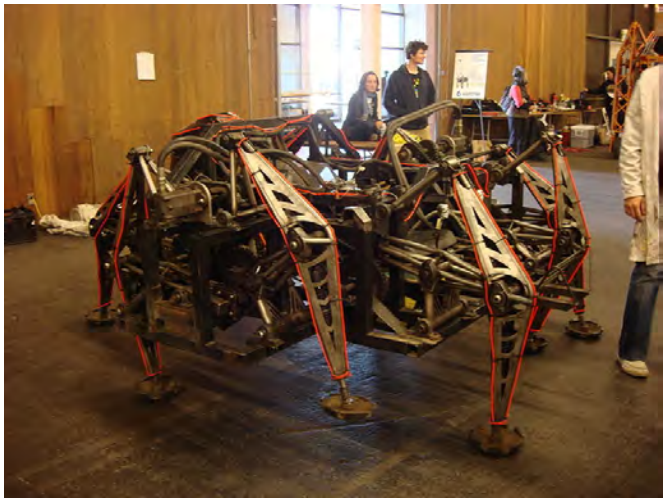
I think that two legs are a disadvantage though, because all of the weight of the robot must be supported by two legs, and when it's walking, only one must support the weight. This is a problem with both stability and power. I really like how his movements are just plucked out of something that works- nature.

This orange robot is one of the most common legged robots- a hexapod in a "dual in line" style. There are two rows of legs, and two lines of symmetry. This is a more advanced hexapod, but the base is essentially a rotating joint to the leg, with a hinge halfway down. This allows the robot to lift up and place the leg, and is a very simple design in comparison to Asimo. It is also generally accepted that 6 legs is a good compromise between weight and complexity.



My favourite ideas from this page are the 3 axis movement of the first robot, the natural design of Asimo and the multitude of legs in the final robot. I think that 8 legs, however, would spread the weight of a robot much more evenly, and I could base it on a circular design, like a spider. This incorporates the ideas of stability, multi directional movement and manoeuvrability from the previous page.

Section 1 - Planning - Similar Products - Octopods and Robotic Arms



This is the Mondo Spider, which is a part art and part engineering creation.

It was build on a large scale so that it could carry a human, as a standard vehicle would.

Although it takes inspiration from nature with the number of legs, design and the movement, it is slow (1.2 metres per second, which isn't a very big percentage of its size), very noisy and not agile in any way, unlike a spider, which excels at all of these traits. It is interesting to note however, the, almost, purely mechanical solution to the leg movement, in the use of the Klann linkage, a single engine and several hydraulic pumps and motors.

This is a very interesting project, however, I am nervous as to how the mechanics of this vehicle will scale down onto a small robot. I am also concerned about how the mechanical solution does not allow for my agility or freedom of movement other than a simple walk.

As there are not many examples of many legged vehicles with much freedom of movement on the legs, I looked at some robotic arms, which are essentially upside down robotic legs. The photo on the right shows a 7 axis robotic arm, however, without including the "hand" or "wrist", it has 3 axis. At the base, there is a hidden servo (inside the circular compartment) which rotates the entire arm. The bottom-most visible servo hinges the whole thing forwards and backwards. The combination of these two axis produces a range of movements identical to a ball and socket, but negates friction and complexity of manufacturing. Further along, the arm breaks to reveal a hinge ("elbow"). This is a simple join.

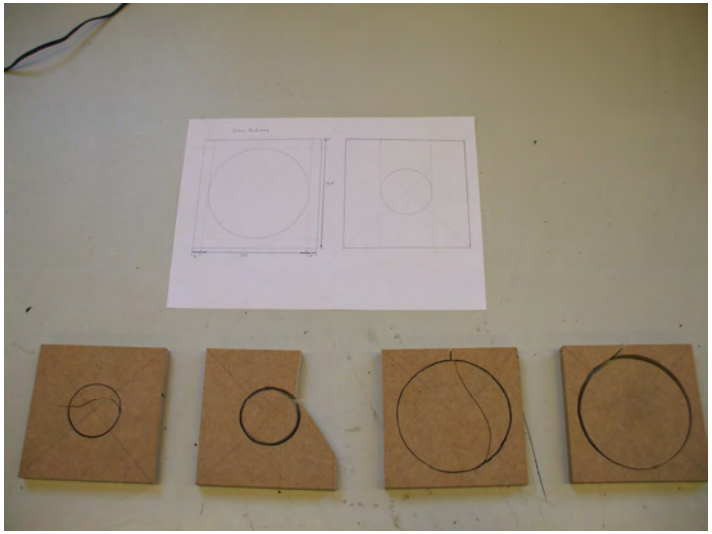
I really like the huge range of movement possible from 3 axis of movement, it is much more than 2 axis, but does not allow much less freedom than a 4, 5, 6 or 7 axis arm (or leg).

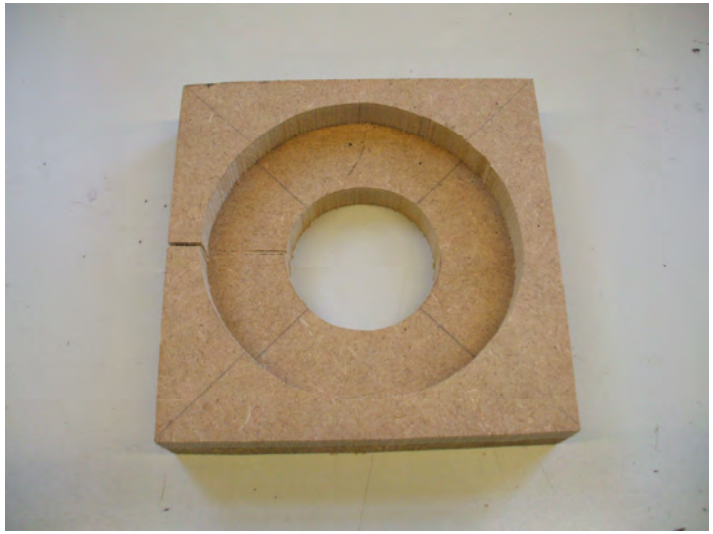
Initial Plan

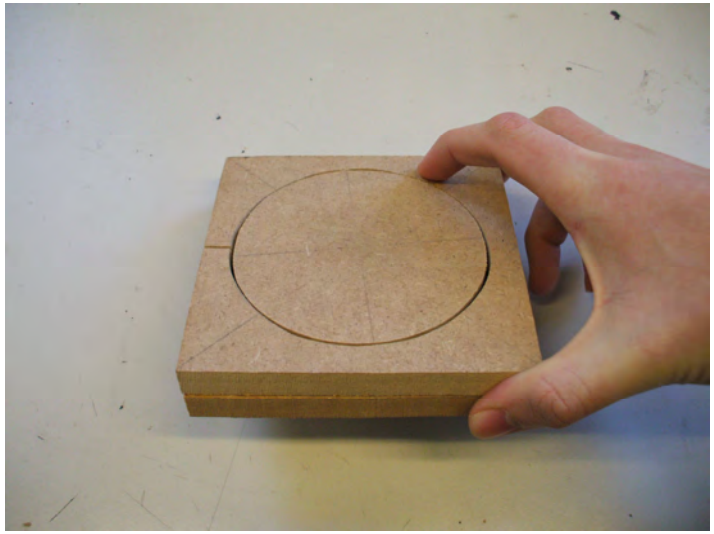
My comparison of all terrain vehicle led me to the conclusion that I wanted to make a vehicle with legs, but is also showed me the importance of stability, multi directional movement and manoeuvrability. My analysis of robots with legs led me to a circular design, looking at nature (spiders), complex leg design and multiple legs. I will build a robot based on a circle, with 8 legs, and 3 axis of movement. This seems to be the most effecticient amount of degrees of freedom for each leg, and it seems to work really well with the robotic arm above (excluding the hand).

Section 2 – Mechanical Research - Bearing Race

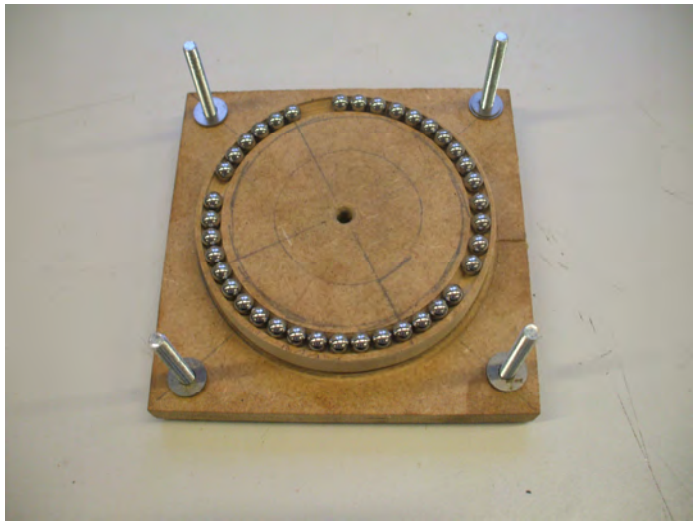
I researched the idea of frictionless (or as near as possible) joints and decided to make a bearing race to test it for affectivity and viability for the base of each leg.

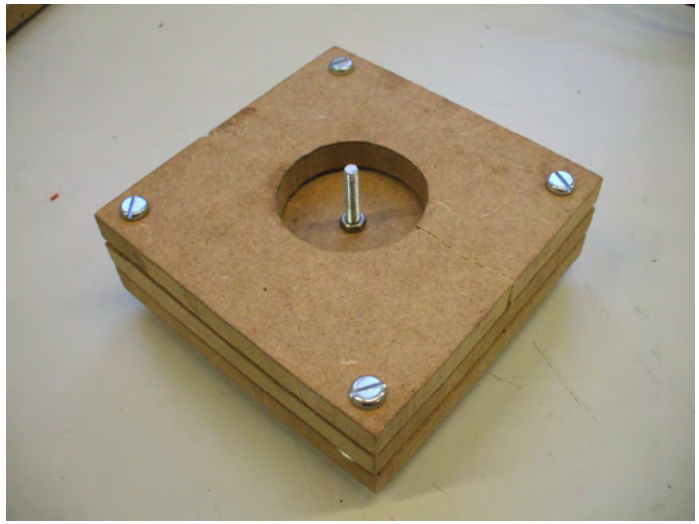






I cut the pieces required on a bandsaw, and glued together the slots I made to get the blade in. I then used a lathe with a U shaped tool to cut grooves into the base and bottom of the turntable. I found that with uneven weight on the platform, it would tend to flip. I therefore made a lid which was identical to the bottom, to hold it in place. I cut grooves in this too, to double bearing race it and places ball bearings in to the grooves. I then bolted it together as hard as possible and used to drill on the centre bolt to loosen it up. I then added appropriate washers between the pieces of wood to decrease friction.





The bearing race is a fairly effective solution, but it is too clunky and would be difficult to make myself on a small scale. I also doubt that it is necessary, and expect there is a better option. Ball bearings are also quite expensive.

Section 2 – Mechanical Research - Comparison of Materials

	Plastics	Wood	Metals
Insulator of electricity			Most conduct or conduct to a degree
Cheap			Compared to the others, quite expensive
Light		Strong woods are very heavy	Some are very light, others very heavy
Easy to shape			Hardest to shape
Can be cast		Does not melt	
Can be injection moulded		Does not melt	
Can be vacuum formed		Does not melt	Too high melting point
Ductility			
Durable	Cracks and chips with use	Swells and chips with use	
Waterproof		Swells in wet conditions	
Resistant to deformation	Only deforms under heat	Deforms in many circumstances	Some metals bend easily
Strength	Very strong for it's weight	Relatively weak	Definitely the strongest of the three
Aesthetics	Completely customisable	Can be painted or varnished	Completely customisable except transparency

I decided to use a combination of metal and plastics for my robot as they complement each other perfectly and provide the right aesthetics. However, because of the cost and ease of use, I will use wood to prototype. I will look at specific examples of plastics, woods and metals later.

Section 2 – Mechanical Research - Bracket Ideas

I hand-drew the diagrams for this section, while I was working on the solutions. These can be found over the next two pages. They include a brief summary of advantages and disadvantages.

For the leg to behave like a real leg, I decided that it had to have a ball and socket joint, followed by a knee joint on the leg. I researched and created myself some ideas to emulate a ball and socket joint.

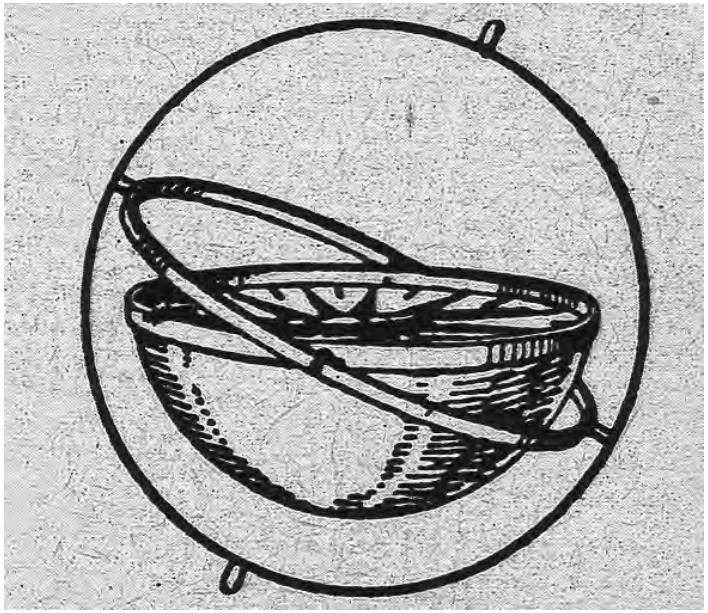
Some notes on some of my ideas:

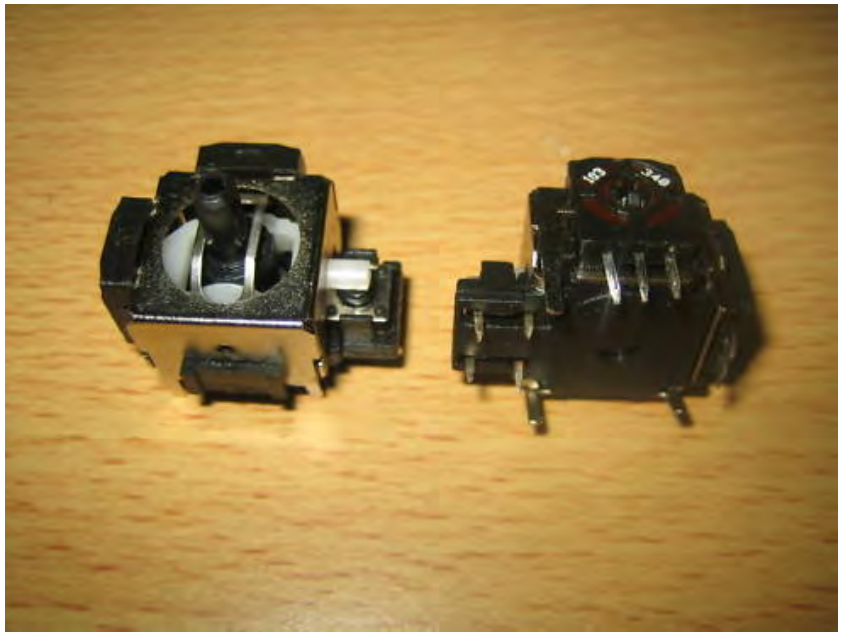
Canfield Joint - Designed for high power applications such as space thrusters. An unrealistic option, but an interesting one nonetheless.

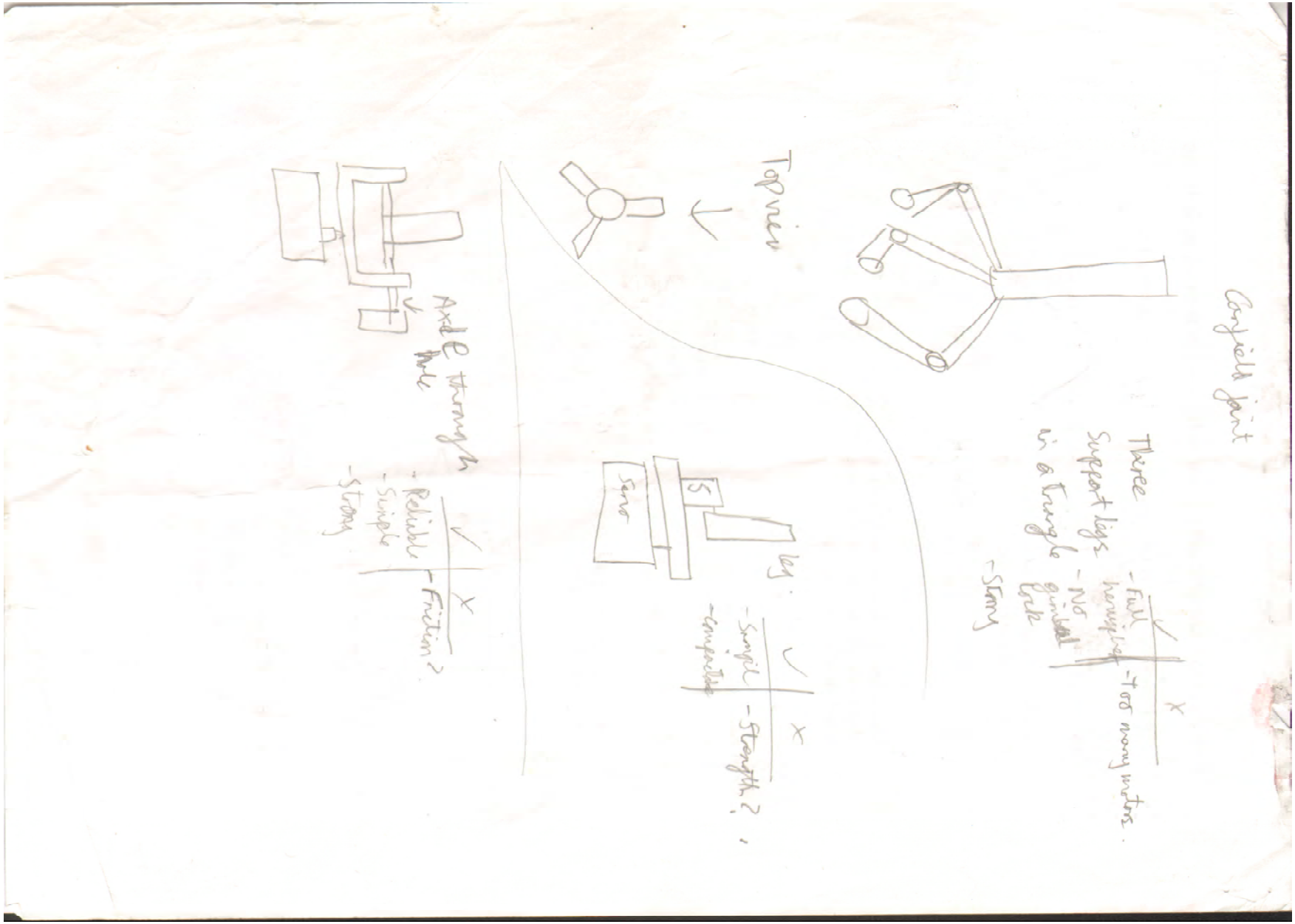
“Axle through bracket” - A design of mine, allows for ball and socket movement about the intersection of the axle and the line of action of the rotation.

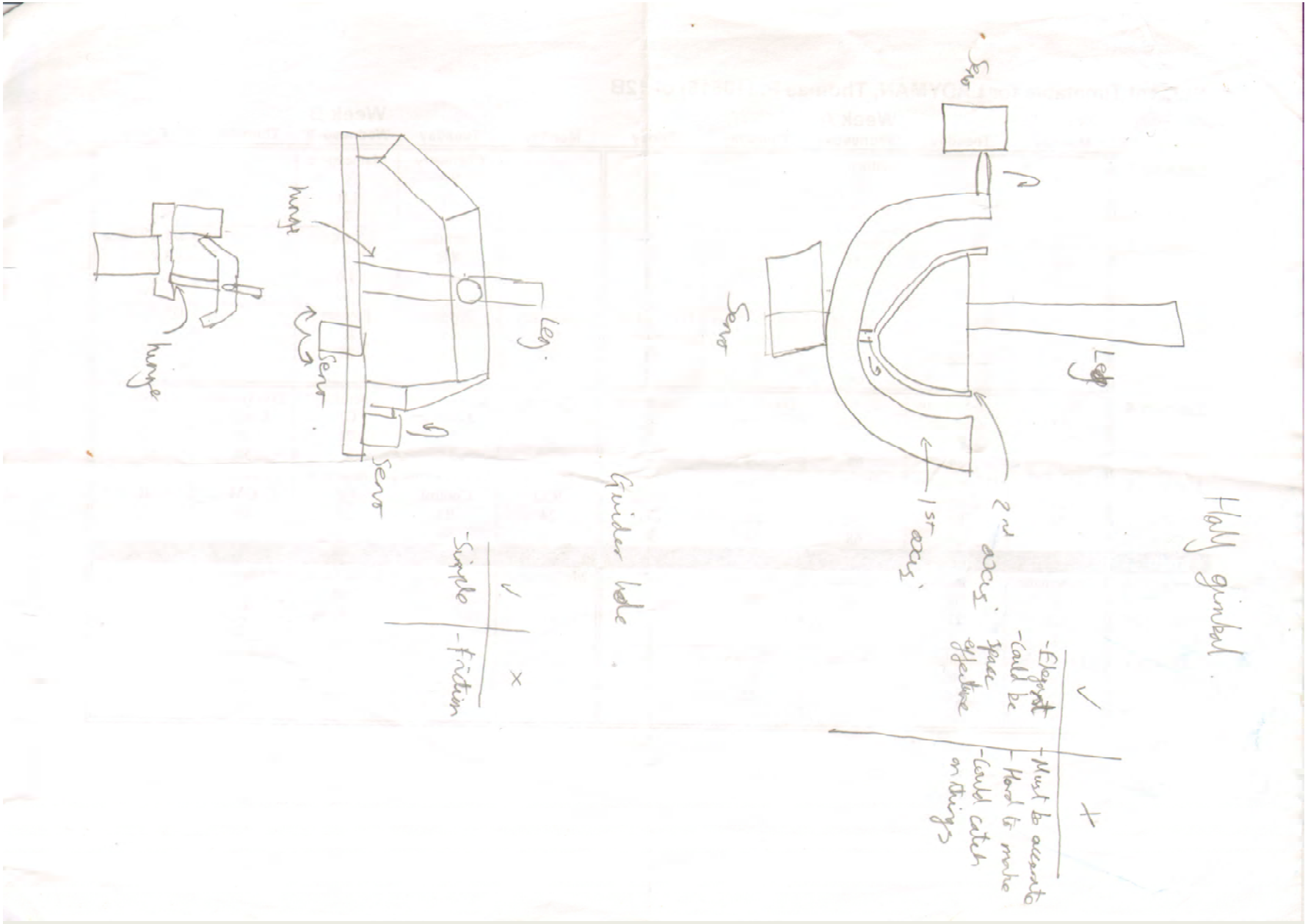
“Half Gimbal” - Another of my ideas, with help from a teacher at school. It is based upon an upside down gimbal (device used to keep ship compasses still). By motorising the axis, instead of making them frictionless, as in a ship, it is possible to create a ball and socket emulation.

The “guider hole” design came from a dismantles joystick which I looked at for ideas.





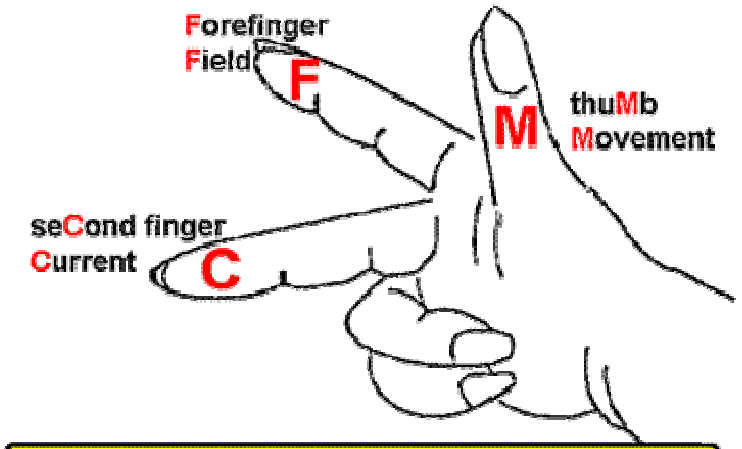

Gayfield joint
Three support legs
in a triangle
- Full freedom
- NO gimbal lock
- Too many motors
- Sturdy
Top view
Servo
Leg
- Simple
- Strength?
Axle through hole
- Reliable
- Simple
- Strong
- Friction?


hinge
hinge
Leg
Servo
Servo
Guider hole
Simple
-friction
Servo
Servo
Leg
2nd axis
1st axis
Holly gimbal
-Elegant
-could be space effective
+Must be accurate
+Hard to make
-Could catch on things

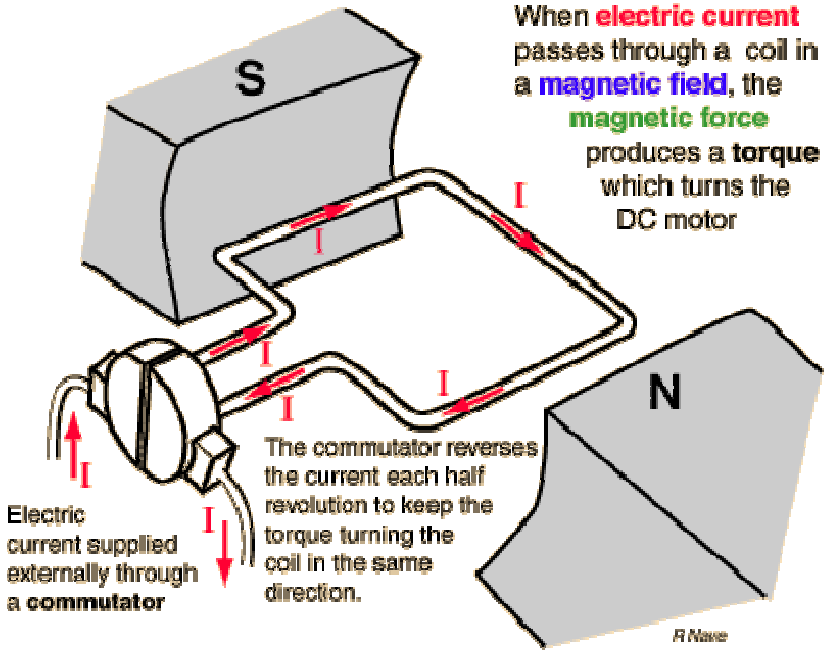
DC Motors

When a charged particle moves, it induces a magnetic field and vice versa. This can be both useful and an annoyance. For example, a bundle of wires wrapped together will all induce their own magnetic fields, and if a charged particle can induce a magnetic field, a magnetic field can induce a charge. The resulting magnetic fields then interfere with the signals in the cables. This is not a problem with say, power lines, but delicate computer signals for example, can be made completely useless via carelessness.

However, this magnetic field can be used to create a movement from an electric current, such as with motors and solenoids or to harness movement as electrical energy, as with generators or dynamos.



Inside a dc motor is a magnet with a north and south pole, a coil, brush and commutator, arranged as in the diagram. The brushes are part of the commutator and are normally made of carbon. These carry the current to the moving coil. It would be impossible to just use standard wires, becase they would tangle and rip off.



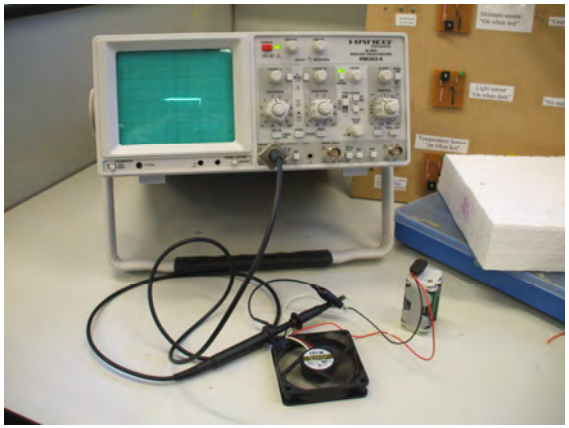
The current induces a magnetic field around the coil (only the direction of the coil is shown in the diagram for simplicity) which is repelled by the stationary magnets. When the coil reaches vertical, there is no resultant turning force on it, but it continues moving due to its momentum.

After vertical, the commutator reverses the current so the resultant force is in the same direction as before. If this didn't happen, the coil, and therefore, axle, would be locked in the vertical position.

In this style of motor, the axle is attached directly to the end of the coil. Other types involve the moving parts being the magnets and the coils stationary. This negates the need for a commutator.

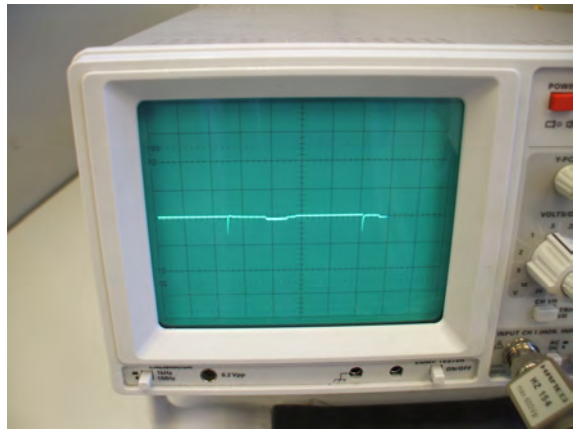
Section 3 – Motors - Encoders

Encoders - Magnet Based

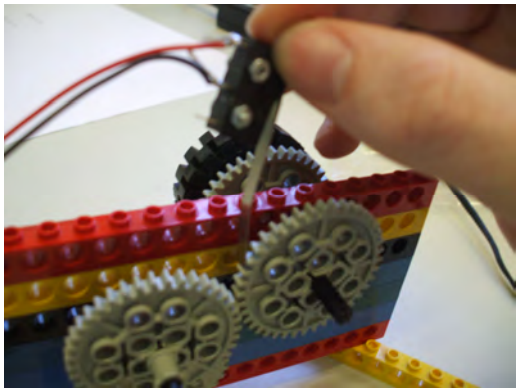


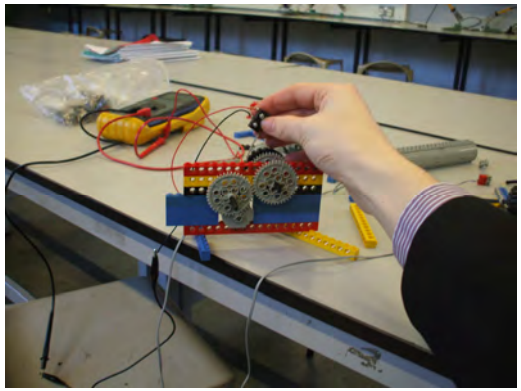
I removed the fan from a computer and attached it to a battery. There is a third wire, other than the two used for power. A magnet attached to the fan inducing a small electric current once every rotation. By counting the pulses, it is possible to determine the RPM of the fan.

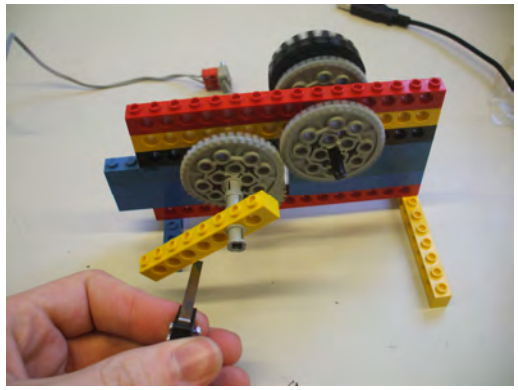
This is a form of encoder as, with a geared motor, it would be possible to calculate the position of the output shaft using the gear ratios.

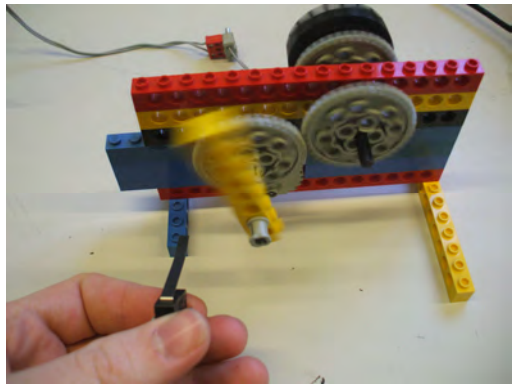


Encoders - Micro switch





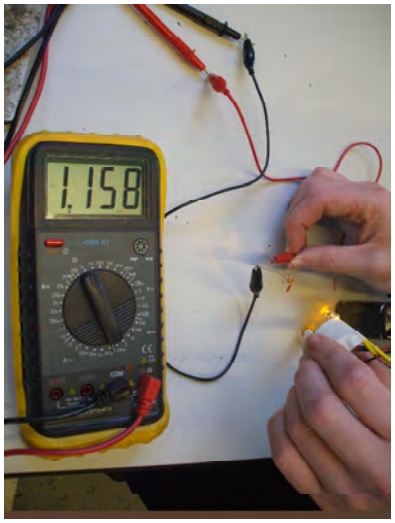




By switching a micro switch using a cog, the position of the cog in relation to its start place can be calculated, if the number of points on the cog is first counted. This can also work for calculating the RPM. The downside is that the switch has to be mounted precisely, so that it is in the right position in relation to the gear.
The same method can be used with a rotating beam, instead of a cog. This can be used to calculate the RPM of the motor shaft, and hence, the position of the motor. It is not really a feasible option because of the extra space required for the beam and the power lost by spinning it.

Section 3 – Motors - Encoders

Encoders - LED/LDR



I taped an LED behind the blades of a motor, and connected and LDR on the other side to a multimeter. The idea was to measure the drop as the blades of the fan blocked the light, however, the LDR was too slow to respond. Also, it would have been highly inaccurate, as the motor only had three blades.

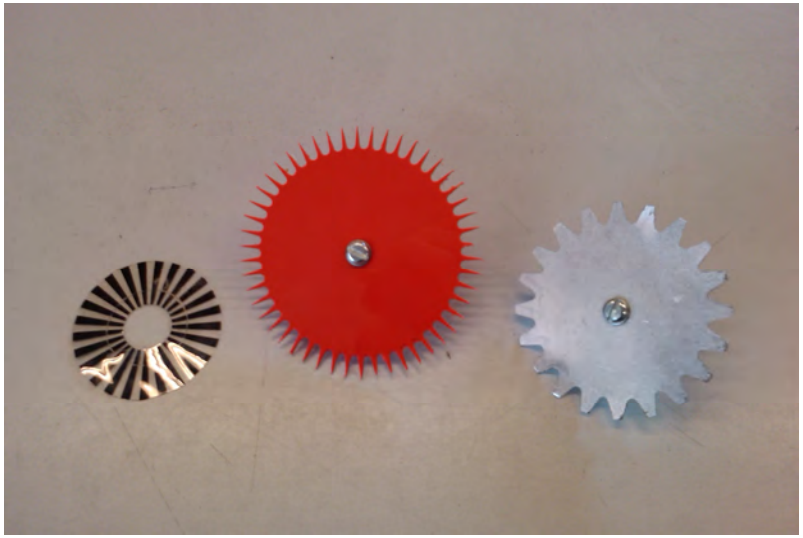
To make light based encoders more accurate, there are discs known as code wheels. Code wheels are circular pieces of materials which can be attached to a shaft. They have “light” and “dark” sections which either block the light or allow it through. This creates a pulse which can be counted.

The code wheel in the centre of the bottom right photo was made by milling a sheet of plastic on an engraving machine. This method produces a wheel with accurately placed “spokes”. The spoke is the dark section, as it blocks the light, and the air space between each spoke is the light section.

Using an engraving machine to make this produces a wheel with a clean finish and spokes which are small but very precise.

With intense light, the plastic is sometimes not enough to fully block it, which is why I made the wheel on the right on the right of the photo.

This wheel is made from a sheet of aluminium, also engraved on an engraving machine. It blocks the light much better than the plastic wheel, although, because aluminium is harder to cut, it has less spokes. This would produce a less precise encoder. Also, it is quite bulky, which could impact on the output power.

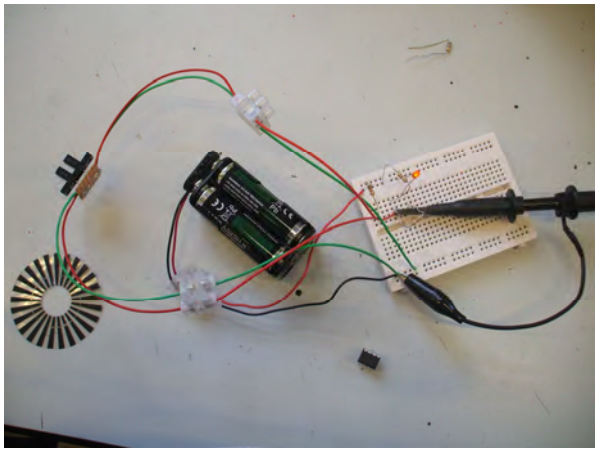


Instead of making the code wheel thicker, reliability can be increased by dimming the light source. After doing so, it is possible to use incredibly thin materials to block the light.

The code wheel on the left is a sheet of acetate which has a code wheel printed on it. The dark sections are formed from the ink on the acetate, and the light sections are where nothing has been printed. Using a program, it is possible to create very precise code wheels, and even multi layer ones. They are easy to make, and very light, which makes the best example of a code wheel.

Section 3 – Motors - Encoders

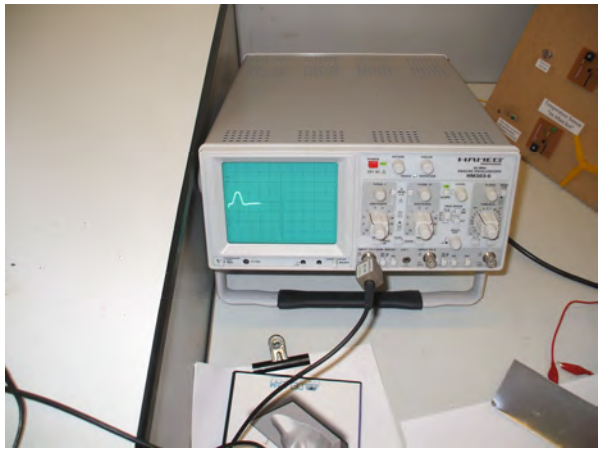
Encoders - Photo switch



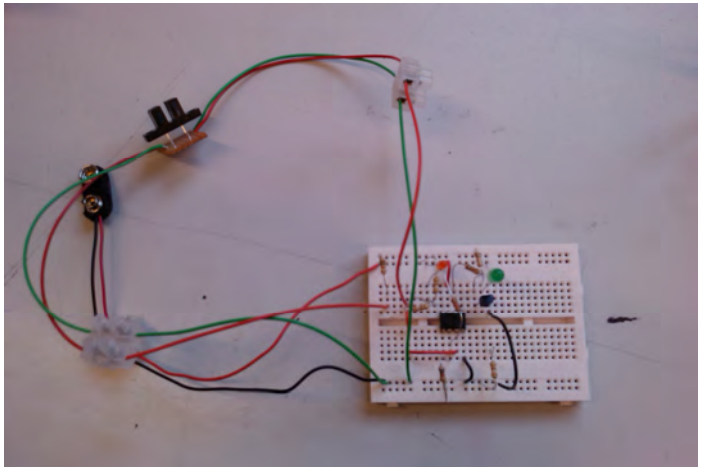
The photo switch is a component in a “U” shape which has a bright infra-red emitted on one side of a slot and on the other side, an infra red receiver. When the emitter is powered, the output of the receiver is high. This means that if an object is passed through the slot, the output pulses.

It is a more accurate form of the LDR/LED encoder.

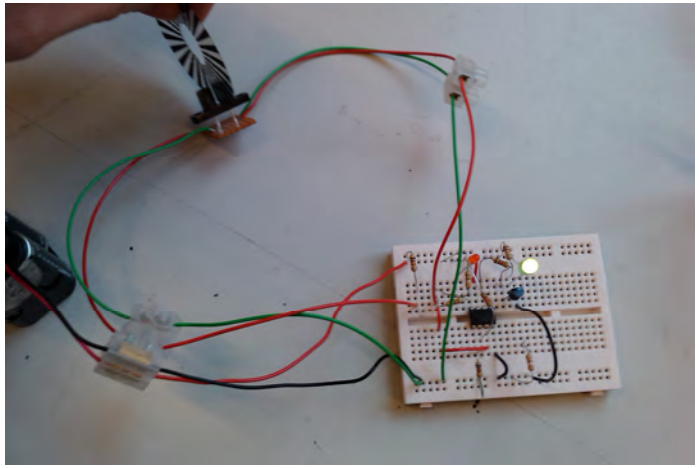
I found that a code wheel printed on acetate wouldn't block the light given off from the emitter, so I used far higher resistors than were necessary to just protect the emitter from current. This dimmed the bulb and made it easier to stop the light.

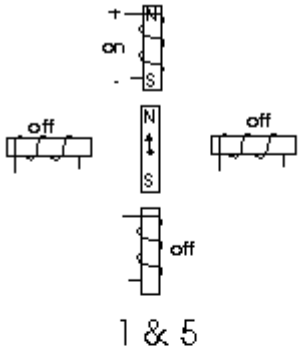


Both the plastic and aluminium encoder discs did block the light as expected, however, the output signal from the switch was very small and needed to be amplified. It was also still, technically, an analogue signal, so I connected an op amp to the output to convert the weak analogue signal to a binary, 0-3v one which could be interpreted by PICs and other components.

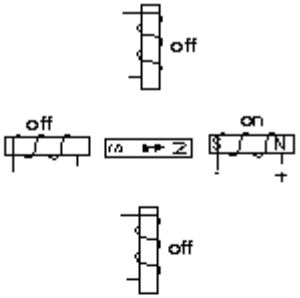


By counting the pulses per second of the op amp using a PIC or a mathematical chip, the speed of rotation can be determined. Also, if the pulses are counted, the position relative to the start can be determined, and the photo switch can be used as an encoder. This is the most accurate way of finding the relative position because the encoder wheel can be so accurately made and the sensor will detect the smallest dark area possible.
The sensor also has mounting holes so it can be positioned easily alongside the motor, whereas the other methods I have discussed use rotating beams or would require gearing to implement.

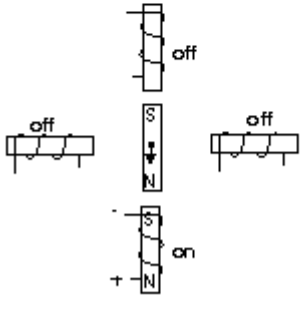




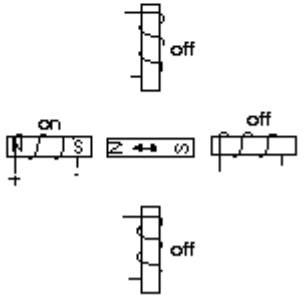
1 & 5



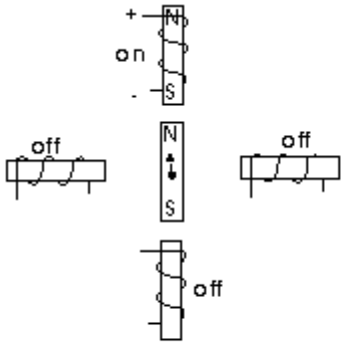
2



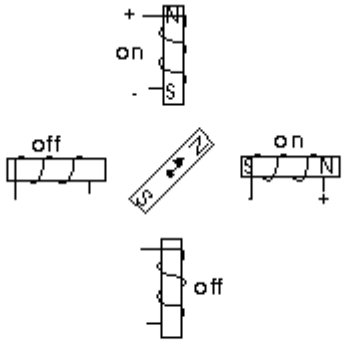
3



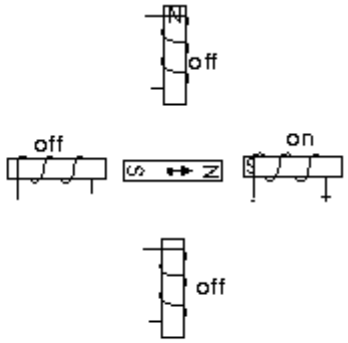
4



I

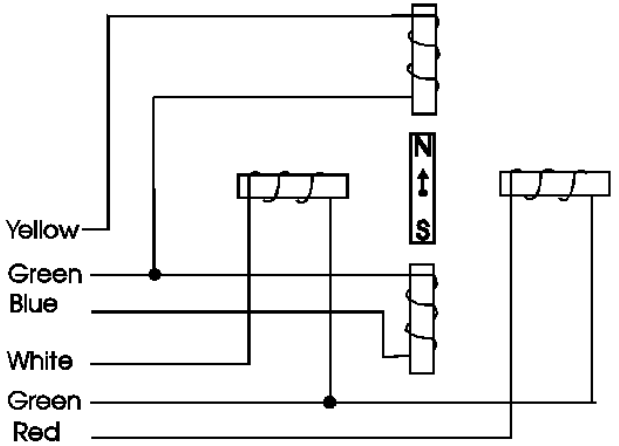


II



III

A stepper motor is a DC motor which moves a step at a time, but in all other respects, is much like a DC motor. Inside, there are 4 interwoven coils, which can be modelled at compass points. If North is on to start with, turning North off and East on will cause the smallest amount of clockwise rotation- one step. By turning them on and off in a clockwise pattern produces a clockwise rotation of 4 steps.



The coils are, however, interwoven, which means that the compass points are repeated until a full rotation occurs.

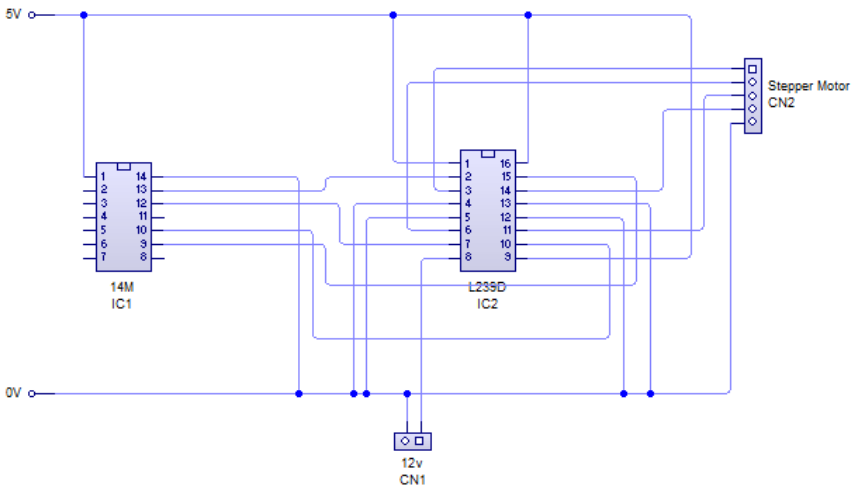
However, instead of powering just one coil at a time, it is possible to power the opposite coil with the opposite charge, to create a torque-ier rotation. Also, to hold the axle stationary, it is possible to positively power say, North and South, and negatively power East and West.

Likewise, by powering coils next to each other, it is possible to create a North-East (etc.) position.

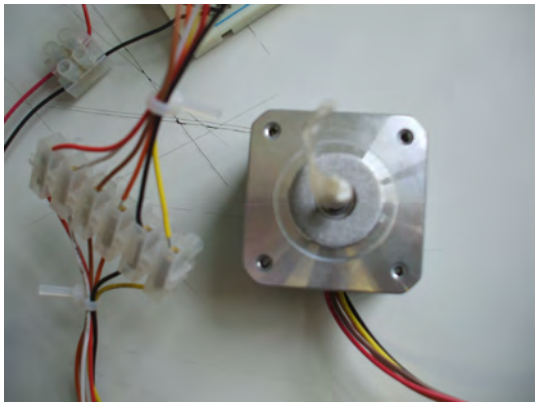
Because the stepper motor can move in discrete steps, these steps can be counted to calculate the position of the axle, however, on its own a stepper motor is an open loop system, ie, there is no feedback to the controller. This means that it is only accurate when the exact load on the axle is calculated before and it is quite a non versatile tool.

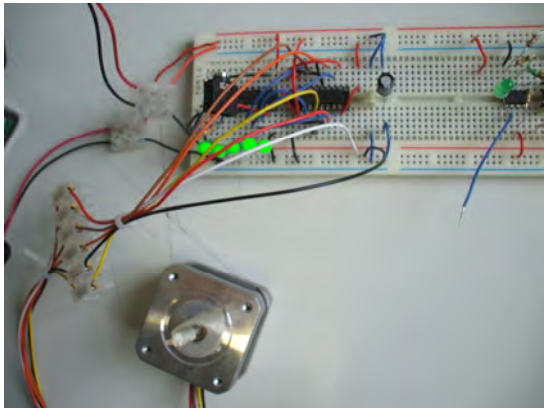
Printers, 3D printers and tools such as milling machines make use of stepper motors for gentle movements, but also use an encoder to accurately position the axle.

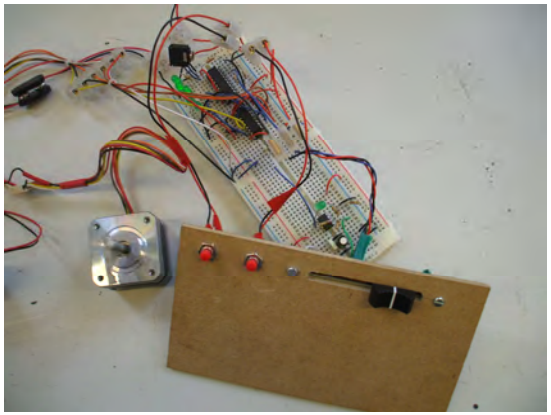
Section 3 – Motors - Stepper Motors



I built this circuit by following a diagram and programming technique in “Programming and Customizing the PICAXE Microcontroller”.
The L239D is a half H driver, which drives the coils as if they were individual motors. This is a layout which allows bidirectional control of two DC motors.
The pin states in the tables below show the sequences needed to make particular movements.
I put LEDs on the output to provide a more visual representation of the output states, however, when the motor is in constant motion, they strobe so fast they appear fully on.
In the bottom right, you can see a controller with a sliding potentiometer and two button, to control the direction of the motor. I used the readadc function to read this value and vary the size of the pauses between changing pin states.







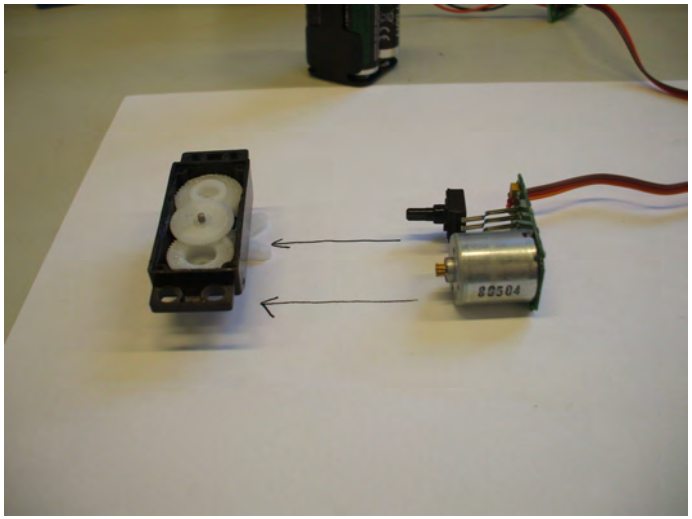
Single Step:	
Forwards:	Backwards:
0001	0001
0010	1000
0100	0100
1000	0010

Single Step High Torque	
Forwards:	Backwards:
0011	0011
0110	1001
1100	1100
1001	0110

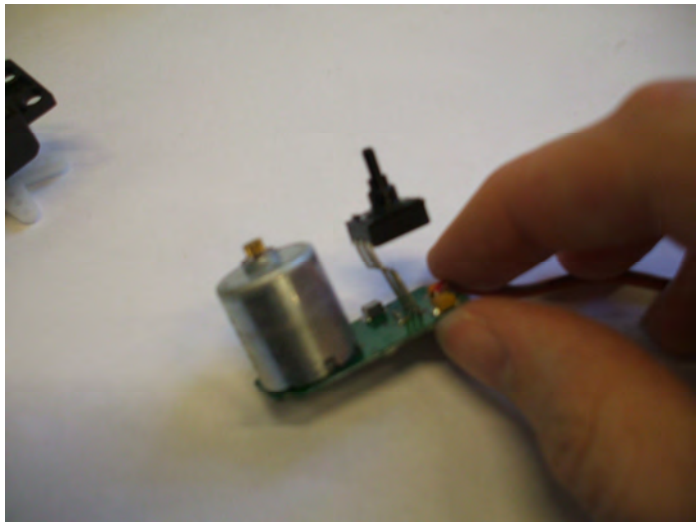
Half-Step	
Forwards:	Backwards:
0001	0001
0011	1001
0010	1000
0110	1100
0100	0100
1100	0110
1000	0010
1001	0011

Section 3 – Motors - Servos

Servos are primarily used in model aircraft to control the flaps and rudders of planes. This means they are light, strong enough to overcome the air resistance, and reliable. Inside a servo, there is a dc motor, gearbox and encoder circuit based on a variable resistor. As the motor shaft turns, it turns the gearbox, which in turn moves the variable resistor. A circuit inside the servo reads the resistance in the variable and controls the motor shaft accordingly. A wave passed into the circuit controls the position. Because they constantly right themselves, they are accurate, however, can sometimes be jittery. The power of a servo depends on the voltage it is run at, so it is important to use near the maximum possible.



There are two kinds of servo, one is analogue and the other digital. This does not refer to the input method, or the output result, as both are identical. Inside the analogue servo, there is a circuit board based on logic gates etc., whereas a digital servo has a PIC inside.

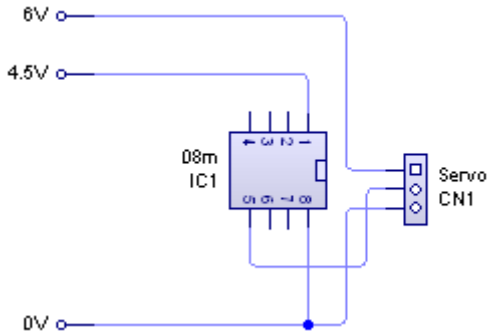


There are disadvantages to both kinds: digital servos are more electronically noisy, and draw slightly more power, whereas they are faster and more accurate than their counterparts.

PICAxe chips have set commands for driving servos, however, the signal is just a square wave, and if the duty cycle is calculated, other commands, such as pulsout can be used instead.

Torque values per distance value are used to measure the strength of a servo (for example, 1.2 Nm/cm), this means that if the movement acts over 10 cm, the strength is a tenth of the specified strength. Horns are attached to the grooved shaft of a servo, to be bolted onto the part needing to be moved. Servos usually come with a selection.

When choosing a servo, it is important to guarantee the accuracy of its positioning (by trying one out or reading reviews on the internet) and it is useful to look for a bearing raced (or double bearing raced) shaft, metal gears (nylon ones strip themselves under load), metal casing (helps with cooling) and whether they are digital or analogue.

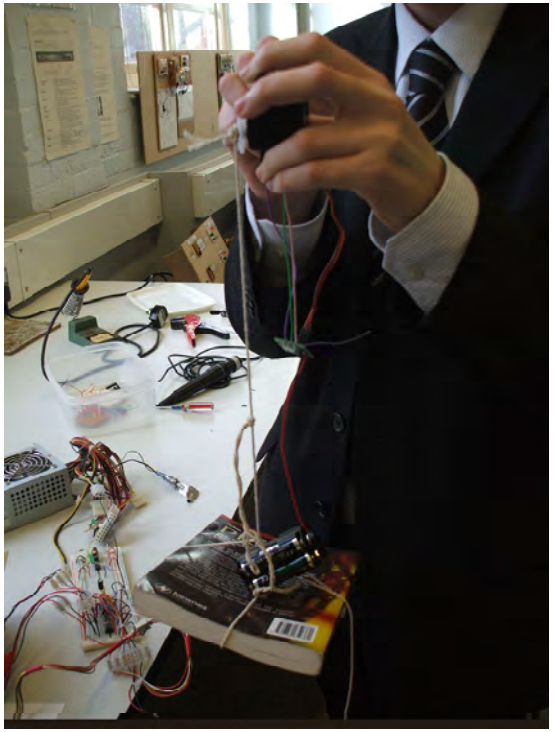


Some servos use a 5 wire connection and others a 3 wire connection. It is important to control the correct pins, to avoid damaging the internal circuit.

Section 3 – Motors - Strain Testing

Motors - Strain Tests

As an arbitrary strain test, to test estimate relative performance of the stepper motor and servo, I tied a piece of string around a small book and lifted them using each motor.



Even with the stepper motor in high torque mode (opposite coils repel), it struggled to lift the book. Although this kind of strain test is highly inaccurate and gives no output result, it serves as a reasonable comparison between the two motors.

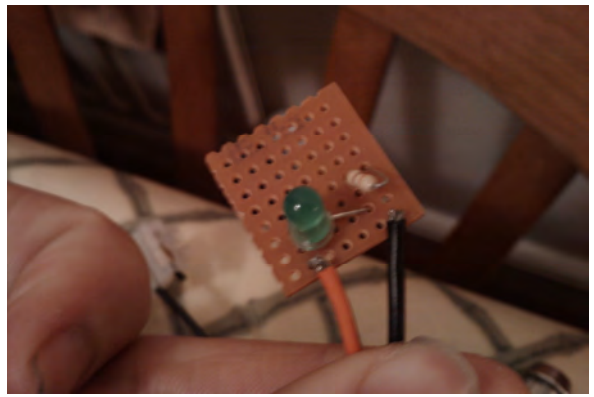
Because servos are comparatively strong, especially for their size and because of their position control, I have decided to use servos for my robotic spider. It will mean that there are far fewer circuits, as extra encoder boards do not need to be produced, saving on space and, more importantly, weight, on the spider.

For testing and demonstration purposes, constant reliance on battery power is uneconomical, ecologically damaging and irritating. I looked into building transformers for mains electricity, but this was too dangerous to undertake without proper training. I therefore decided to use a prebuilt transformer, possibly from a mobile phone. These are the most commonly used mains power supplies for hobby projects, as many people have two or three spares and they're very simple to use.

However, they cannot deliver the kind of power I would need. With 24 servos, each peaking at just less than one Amp, requiring 5-6V, I would need a 5*24=120W (P=IV) power supply, minimum. This kind of power is readily supplied to computers, so I took the power supply unit from an old computer. They are still fairly cheap to buy new though.

To use it outside of a computer shell though, a few modifications have to be made. A switch needs to be added to the switch line (the colour which is not labelled with a voltage), connecting it to a ground line. This takes the place of the computers on switch. I used a key switch to make sure people couldn't fiddle with it when I was not around.

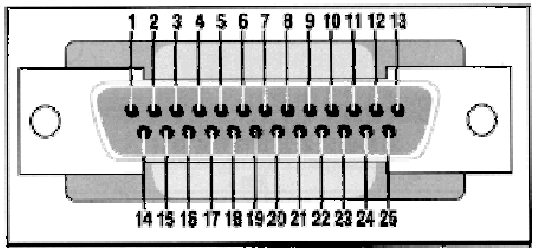
Testing the unit now will result in no output- it needs a load to start supplying power. I put an LED and resistor on some stripboard across a power line and the unit was fully operational.



It is very important that shorts across the supply are avoided, as this unit can supply 20 Amps, and will melt the wire, possibly causing a fire. I found that my particular unit has a cut off if this does happen, but damage can still occur before this cut off happens.

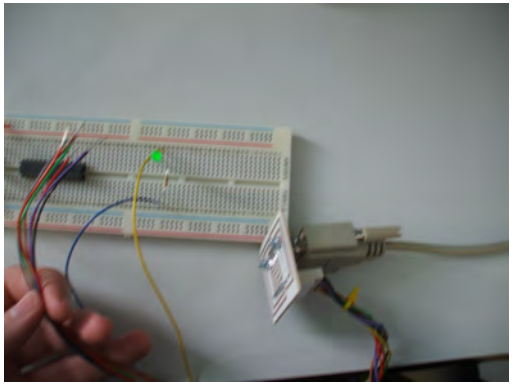
Section 3 – Computer Communication - Serial Port

The serial port used to be the main standard for all devices attaching to a computer. Serial port is actually the incorrect name for what it is associated with. A standard desktop computer will have 2 serial connections, one a COM port (the 9 pin, standard port) and a parallel port (LPT). The COM port is used for most devices (mice, joysticks etc.) while the LPT port was the standard for printers for a long time. Both ports use the RS-232 standard, which defines serial communications.



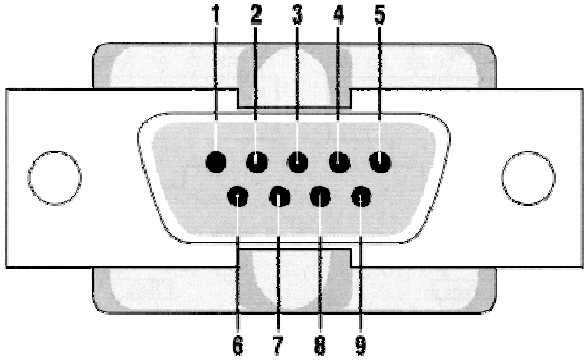
Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CH	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

RS-232 uses flow control, which involves the manipulation of pin states to show that a device is ready for data, not ready, or requesting to send data. This is the reason that so many pins are used, however, it means that the receive pins don't have to be waiting for data all the time, and helps to prevent corruption through early or late transit.



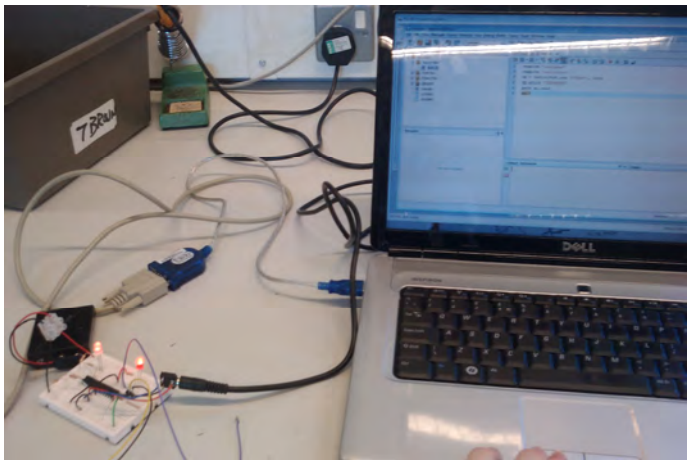
The serial ports should have a 12 Volt supply, however, USB to serial converters bypass this by providing -5V to +5V which gives a range of 10V- almost 12.

The photo on the left shows a serial board I made which converts to the pins to a header. The LED is being directly controlled by the serial port. The unused wires are flow control, which I ignored.

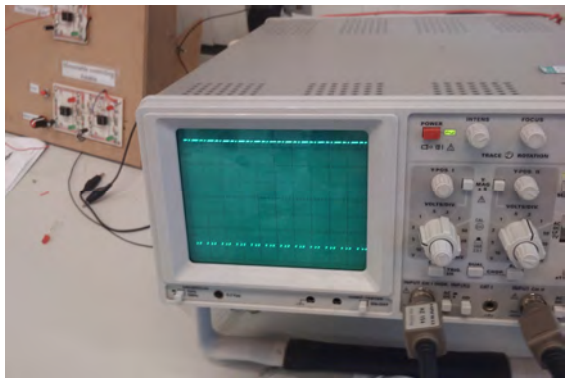


Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

The photo in below shows a more complex version of this, with a PIC validating the serial stream, and turning on an LED when the right data is sent.

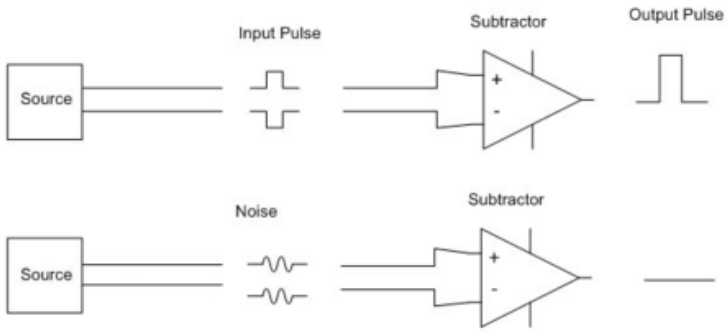


The computer program was written in Ruby with the Serial-Port extension.



Section 3 – Computer Communication - USB

USB stands for Universal Serial Bus and is the most commonly used computer peripheral, with devices ranging from mice to oscilloscopes. It provides a 5V supply, and has two data pins, which are used in an unconventional way. Whereas protocols such as UART (serial) use input and output lines, coupled with flow control, USB has two data pins, but effectively one data line. This is because a major problem with serial ports is noise around the computer- USB was specifically designed to overcome this. The two pin wires are twisted together to form a twisted pair.

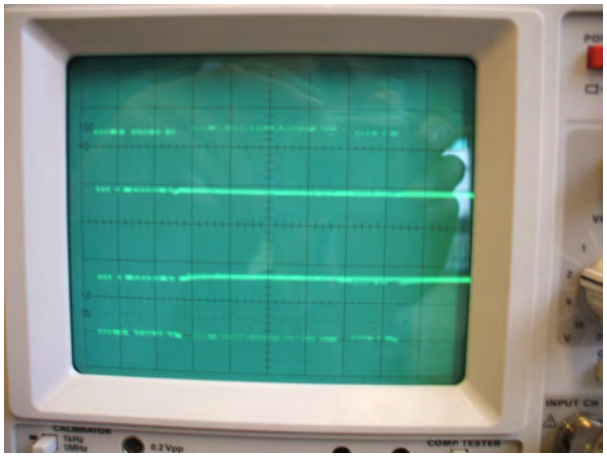


The data line is bidirectional, pulled high or low inside the computer or the device at the other end. By sending the data on one line (D+) and the inverse of the data on the other line (D-), both signals will add to zero. However, if you subtract the negative data from the positive data, the result is a higher peak of the original data with common areas (ie, noise, as it is a twisted pair) completely ignored.

This technique is used in combination with a grounded metallic sheath to stop virtually all noise from interfering with the signal.

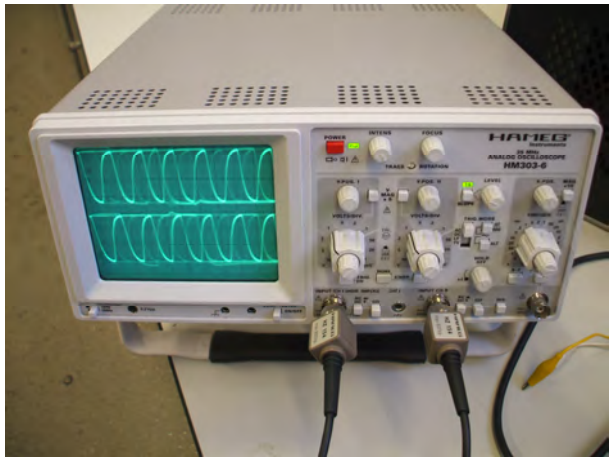
The other major benefit of USB is the speed. USB 2.0 allows for speeds of 480 Mbit/s and USB 3.0 is 10 times this.

However, because of these two advanced features, it is very complicated to use. It is incredibly difficult to interface it directly with a microcontroller and a differential amplifier because of the speed the incoming signal has to be processed, as well as the complexities of programming the signal to be emitted from the computer.



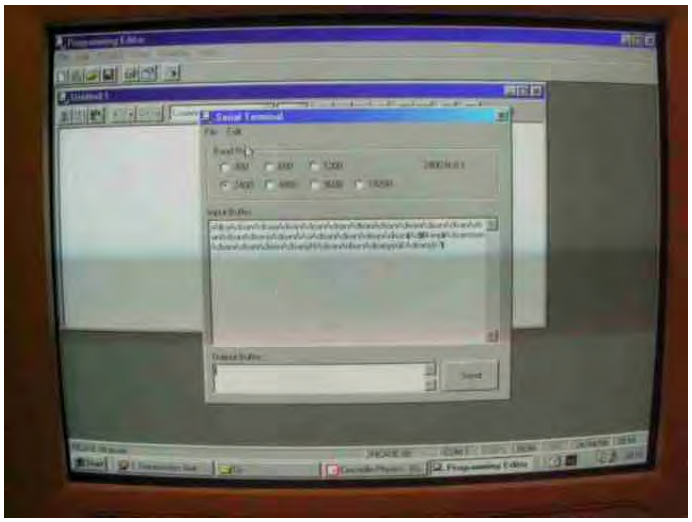
FTDI produce very popular chips which handle the USB protocol and output a UART signal. These are used, for example, in the PICAxe 027 download cable.

I used an oscilloscope to look at the wave produced from a USB mouse and it's signal on a diff amp. The photos show unclear waves because the data is far too fast for the oscilloscope to display properly.



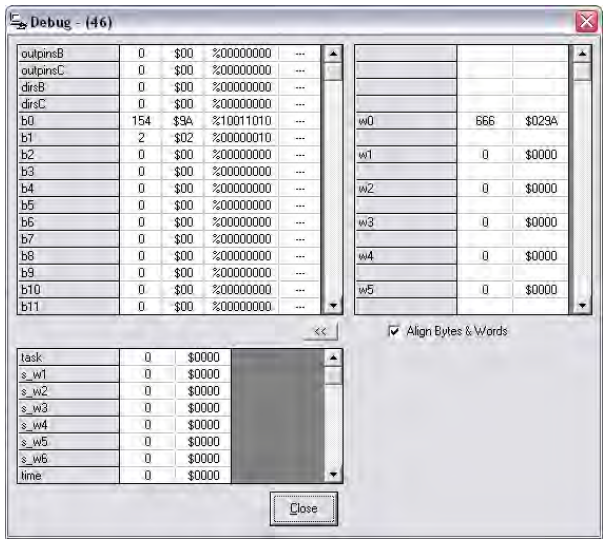
Light emitting diodes (LEDs) are great to provide an instant, simple feedback to the user. They are by far the simplest output to use, as well as the cheapest. They can also be arranged in matrixes to allow lots of them to be run from few outputs (discussed later). Also, with multiple LEDs, bars or LED displays, it is possible to display letters.

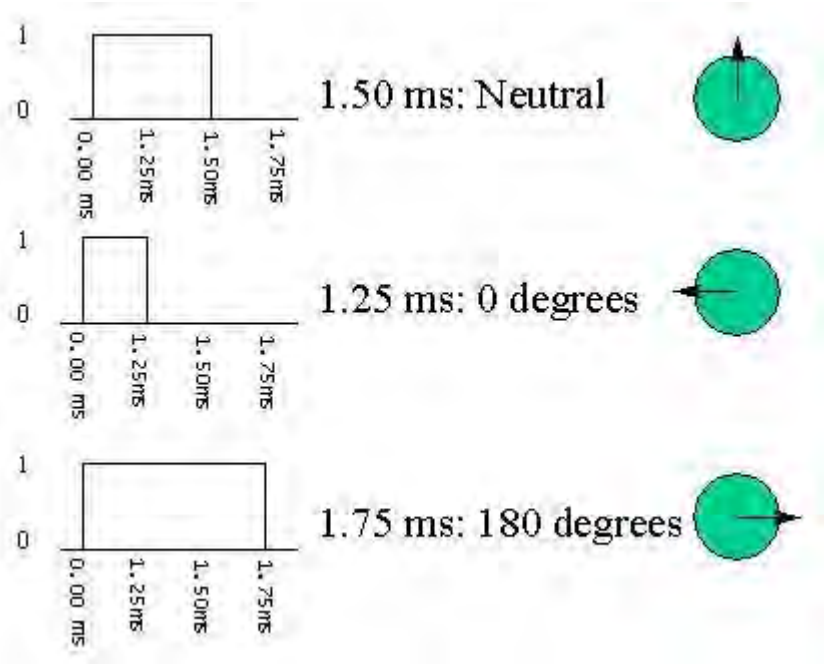
Liquid crystal displays (LCDs) have at least two inputs (clock and data) and are an intuitive way to display rapidly changing information. They can also be used to scroll through settings, greet users or display instructions. They require no computer to be present but are limited by the size of the screen. They also cannot be seen in darkness, without a backlight, and are easily damaged (the liquid crystal often leaks).



Computer terminals are the most versatile tool I have found to communicate with electronics, however, they are not appropriate for an end user, merely for development and testing. They can display any information that is sent back from the microcontroller which makes them incredibly versatile. The inbuilt PICAxe tools also allow debugging of the microcontroller variables, which is very useful.

I will use a combination of LEDs and the computer terminal to display information. This is because I feel the interface will be simple enough to have only LED feedback. However, I will attach headers to allow for LCD screen expansion, in case I feel it is necessary.





After powering a servo with between 4.5 and 6 volts, a signal must be passed into it to determine to position it should move to. The signal must pass through a 330R resistor or the internal circuit won't respond to it.

The position is an absolute one, as opposed to a relative one, which makes it completely different to a DC motor. To position such a motor, it must be rotated until it reaches the desired position however, with a servo, the same pulse is used for each position, wherever the start position is.

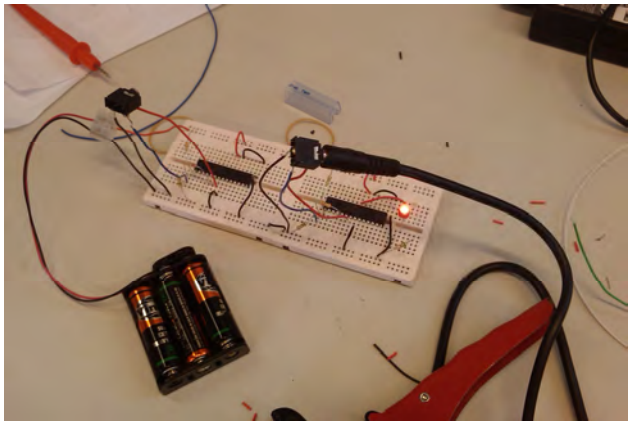
The diagram on the left shows the time period of the signal required to position the shaft. They aren't quantised waves, however. This means that a time period of 1.25 microseconds would be perfectly acceptable.

Square waves produce jerky movements with servos, but it is possible to gradually increase the time period to create smooth, lifelike movements. This requires many more processing resources though.

PICAxe chips also have a built in command to position servos (servopos). It can only be used on PORTB and must be initialised with the servo command. It relies on the internal hardware timer, which means that it is impossible to use with other timer intense commands.

Another feature of the PICAxe chips is the pulsout command, which can produce a pulse of the required time period. This achieves exactly the same as the servopos command, but must be updated every so often to stop the servo losing power.

Section 3 – Prototyping - I2c



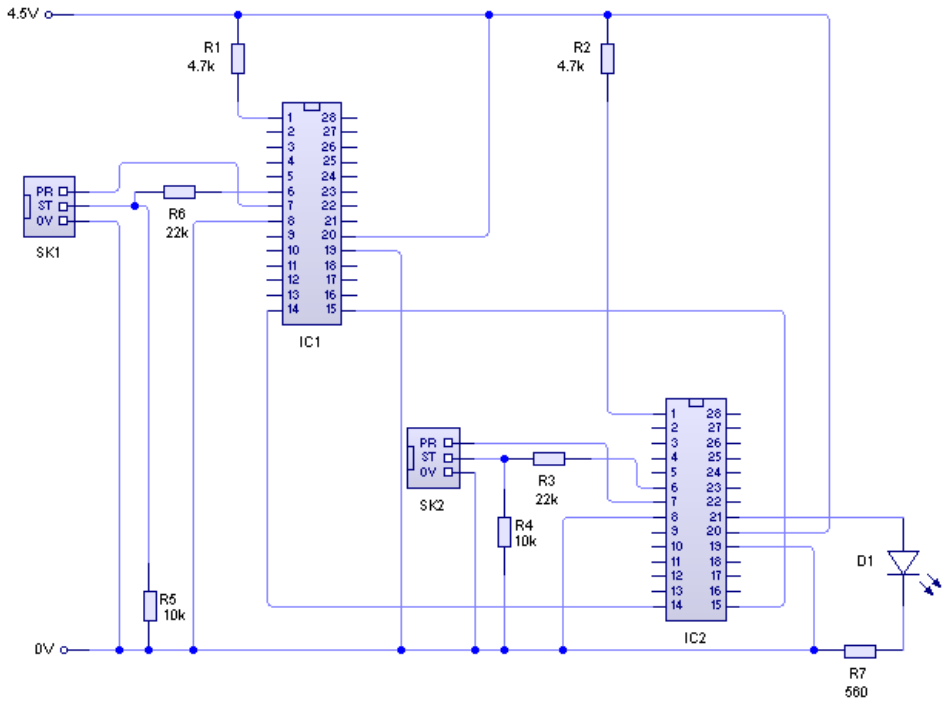
The I2c (inter-integrated circuit) bus was invented by Philips and is a system for bidirectional communication between integrated circuits. It uses two lines, both pulled up with resistors (the value is unimportant, but should be equal), SCL and SDA, clock and data, respectively.

One bus can support one master and up to 112 nodes by default, however, there are more advanced variations of the original version.

The more advanced PICAxe chips have I2c support built in, with specific commands and pins allocated to this function. It is very easy to use.

I constructed a test circuit with two 28X1 chips, one connected to an LED and the other controlling it through the i2c bus.

The chips do not have a default slave address, so I set them to both use an EEPROM address. It's main advantage over say, serial transfer, is accuracy, because of the shared clock, and speed. It is standard for i2c connections to reach 100kbit/s.



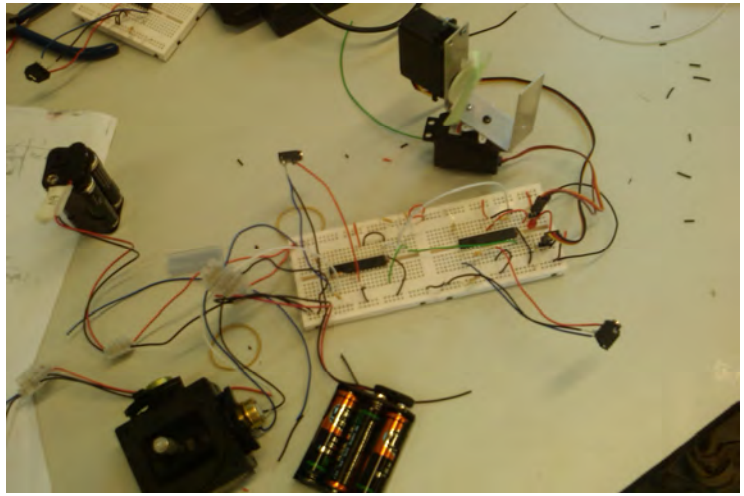
Master:

```
1   init:
2   hi2csetup i2cmaster, %10100000, i2cslow, i2cbyte
3   main:
4   b1 = 1
5   hi2cout 0,(b1)
6   wait 1
7   b1 = 0
8   hi2cout 0,(b1)
9   wait 1
10  goto main
```

Slave:

```
1   init:
2   hi2csetup i2cslave, %10100000
3   goto main
4   main:
5   if hi2cflag = 0 then main
6   hi2cflag = 0
7   get hi2clast,b1
8   let outpins = b1
9   goto main
```

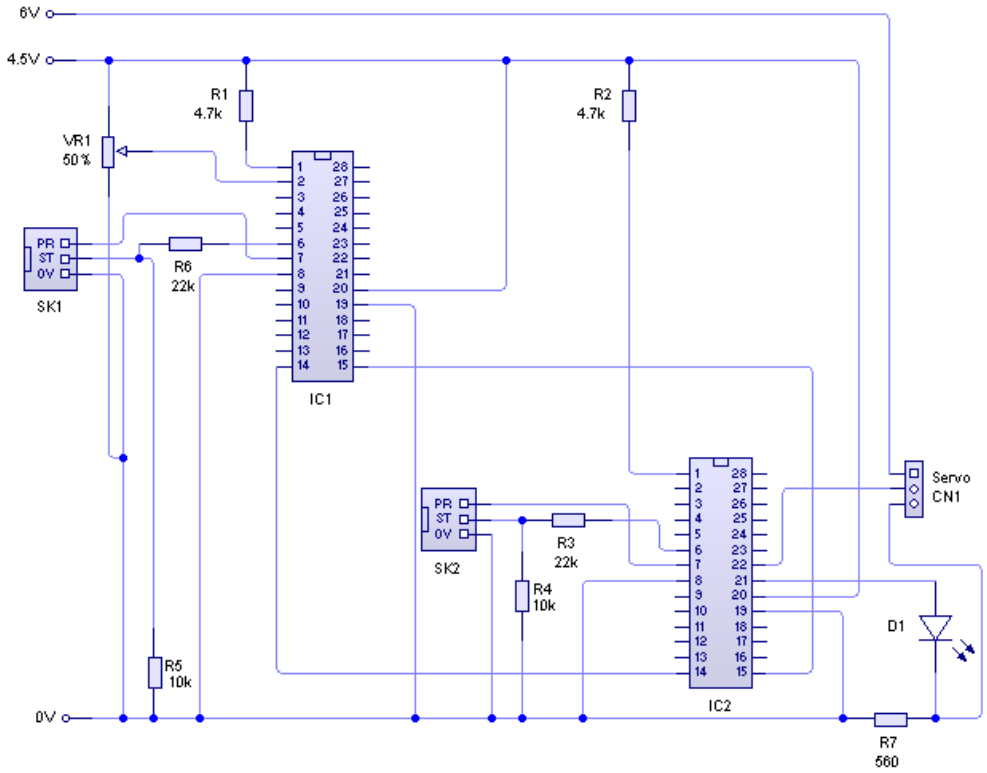
Section 3 – Prototyping—I2c



This is a slightly more advanced version of the i2c circuit on the previous page. Most of the circuit remains the same, except for a variable resistor (joystick in the photo) on IC1 and the servo on IC2.

On each cycle of the first chip, the value of the joystick is read into b0, passed to b0 and sent to the i2c bus.

IC2 sets up the servo for movement and then waits for the data to be sent and adjusts the position of the servo. The last byte sent over the bus is used, as only one byte is being sent. It would be necessary to split the data when sending more complex data, as the last byte would change at 100kbit/s while the chip processes the last few lines. In this situation, a data loss would occur.



Master:

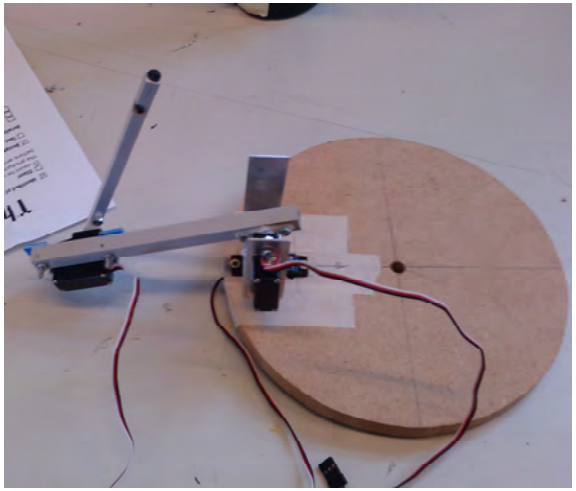
```
1   init:
2   hi2csetup i2cmaster, %10100000, i2cslow, i2cbyte
3   main:
4   readadc 0, b0
5   b1 = b0
6   hi2cout 0,(b1)
7   goto main
```

Slave:

```
1   init:
2   hi2csetup i2cslave, %10100000
3   servo 1, 100
4   goto main
5   main:
6   if hi2cflag = 0 then main
7   hi2cflag = 0
8   get hi2clast,b1
9   servopos 1, b1
10  goto main
```

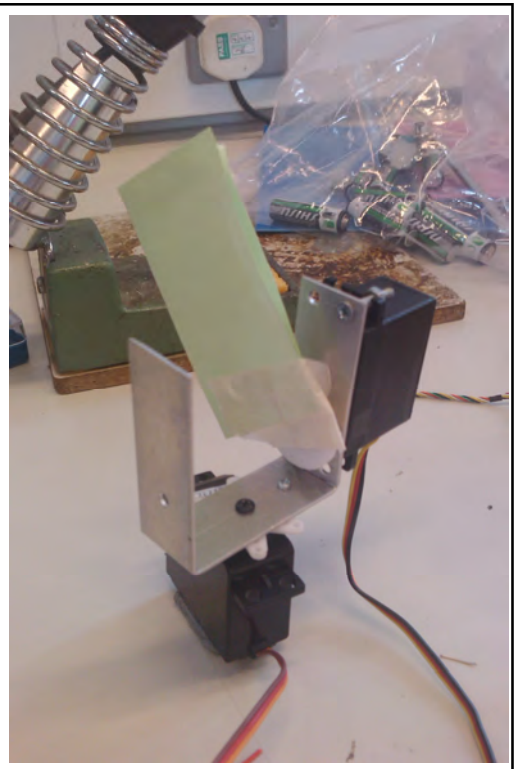
I decided to use this method to control the slave chip, as it was fast, easy and the feature were built into the chips. It also allowed me to choose a slave PIC or a servo driver and would mean that I could extend the functionality of the control bus later on in the project.

Section 3 – Prototyping—Testing the Chassis and Legs

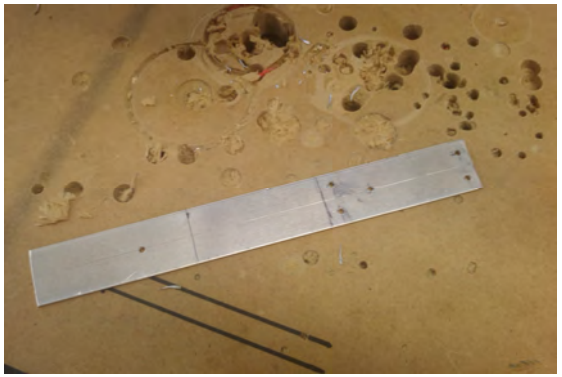


I made up a bracket following my design and tested using a piece of paper. I immediately realised that the bracket was far too wide, and made sure to make the next one thinner. However, this did not test the hinge joint, so I made up a full leg and tested it.

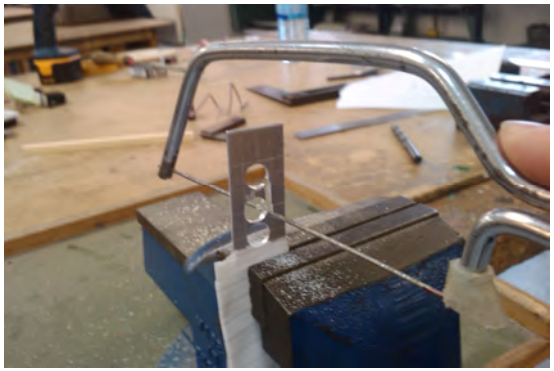
I like the way it worked and, although hard to control, it had a very useful range of movements. It was possible to tape a screw driver to the end and accurately align it with points on the upside down chassis, much like a pick and place arm.

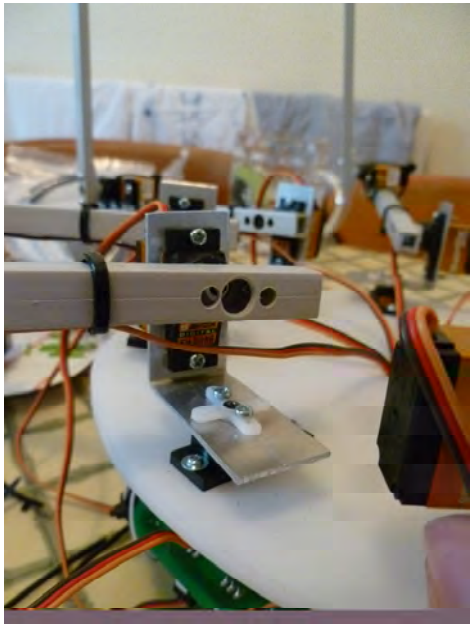


I liked the way it worked, so I made 7 more, to have the full set. Each bracket started with a piece of aluminium strip, which I marked out, drilled the holes for the servo horns, and where the servos would fix. I then drilled out a large section of the square hole and filed it down to the right size. I also experimented with tension saws (abra files) but found filing more effective). I then checked everything fit and bent it using a metal bender.



I also experimented by taping the strips together, and filing them as one. This was much faster and more like an industrial process.



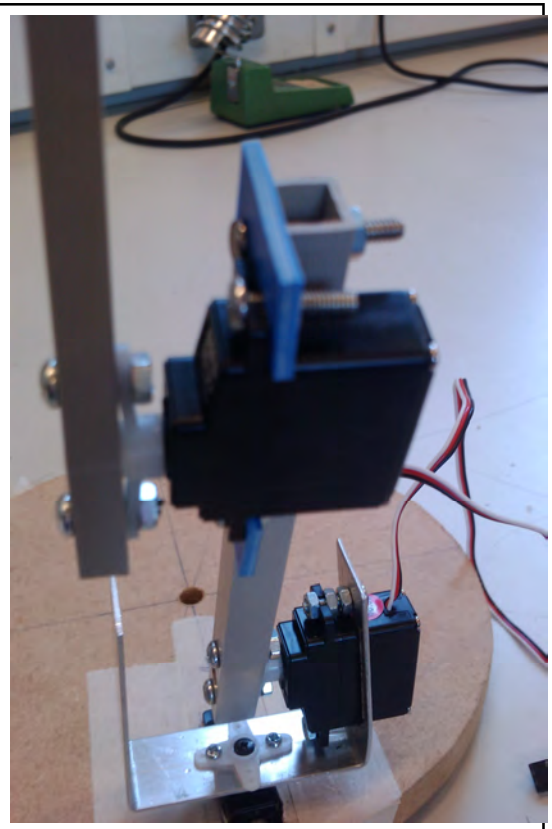


The actual legs were the most complicated and time consuming part of the mechanical building. This is because all the screws had to be accessible, the materials couldn't be weakened, and they needed to be made to an accurate specification.

I started by cutting down two pieces of plastic box section, to the right length. Using two widths, I could have a thicker upper leg than lower leg. I then bolted the first servo horn to one end of the upper leg. This would attach directly to the bracket. To make the bolt heads, and the central servo screw, accessible, I drilled holes on the other side of the beam which were large enough to fit both the screw head, and a screw driver through. This made the assembly, disassembly and calibration much easier.

Then, I attached the second servo horn to the lower leg in the same fashion, with larger holes on the other side. This was tricky because this plastic had a much narrower width, so I had to do it carefully. The servo that slots into that horn was the join between the two legs. To attach this to the leg, I made some small plastic brackets, which were just a strip of plastic with two holes. The last step was to drill two holes in the upper leg for these to bolt onto.

Finally, I strapped the joint servo down with cable ties, to make it extra secure.



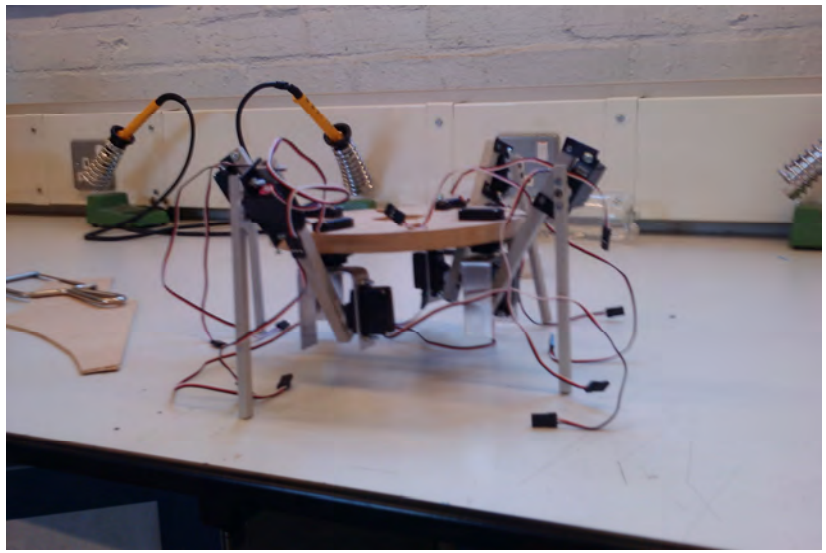
Section 3 – Prototyping—Testing the Chassis and Legs

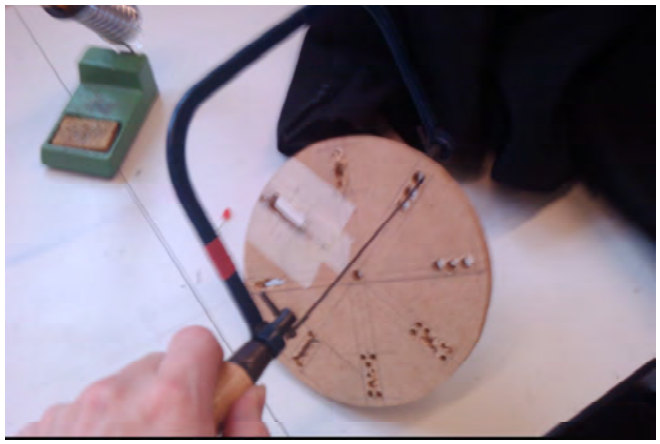
I decided to do very little planning for the chassis, instead to use MDF as a test run to see how my initial thoughts worked and to use as a base to improve on, so I started by cutting out a circle on the bandsaw of 11mm MDF.

The leg brackets I have already designed require the base servo to be mounted upside down, and so I had to make 8 square holes in the base for the servos to fit into and allow room for screw holes. I tested different ways of doing this- using a milling machine, drilling and finishing with a coping saw and drilling and finishing by filing. I found filing to be the most effective, as the other options really needed to be filed at the end anyway, and it was much easier to get a good fit. I left half a centimetre either side of the hole for the screws to go in.
After taking out the 8 holes, I needed to drill a centre hole which was big enough for the servo leads on the underside to fit through. Using a 38mm drill bit, I drilled this hole.

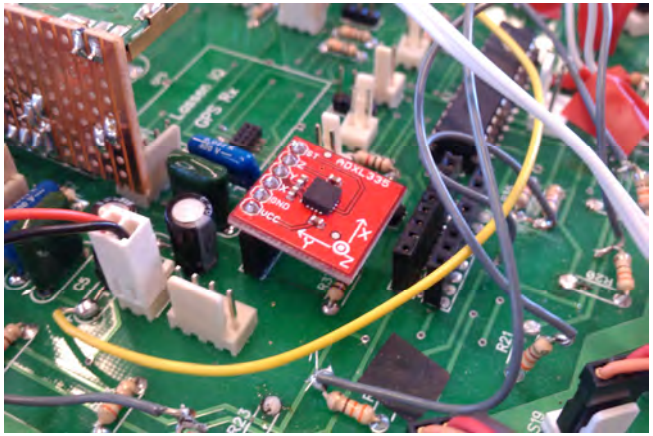
Having already made 8 legs to my design, I started to attach them, and realised that I had not left room for them to rotate fully. I therefore only put 4 on for this prototype.

I found that the wood was very thick compared to the weight it would need to take, and so decided to use a thinner material for the final chassis. The hole in the centre was also much bigger than it needed to be, the legs didn't all fit properly and MDF was very aesthetically displeasing. I noted this for my next version.



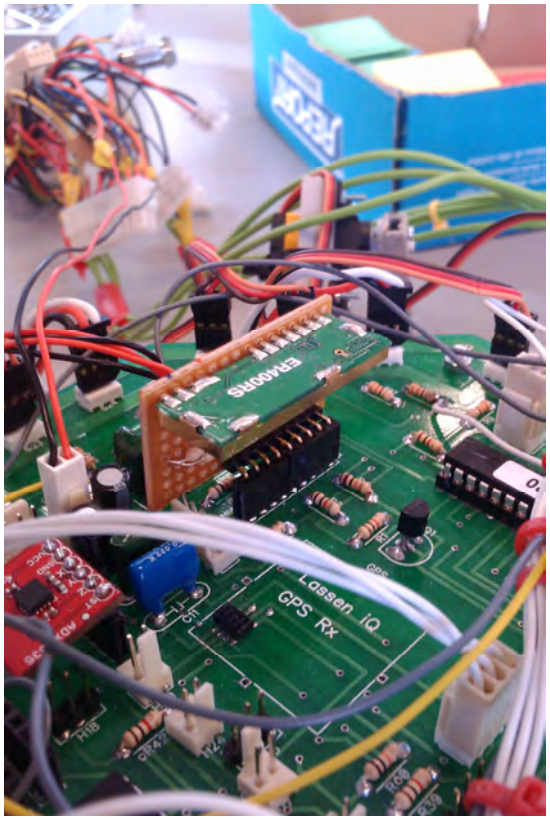


Accelerometers:



I realised that one of the major problems in remotely controlling a vehicle that isn't within sight is determining it's position in relation to the ground, ie, if it has fallen over, or on a steep gradient. I added a 3 axis accelerometer board to my project to provide acceleration based feedback on the position of the robot. By logging and monitoring this, it is possible to accurately show the position of the accelerometer (and therefore, robot) in relation to where it started.

The accelerometer is a package on a breakout board. Inside the chip is a thin piece of semiconductor which is allowed to flex. When the robot accelerates, the bend of the thin strip is measured, and compared to calibrated tables of data to produce a reading. As it is a 3 axis chip, there are three of these. It produces an analogue signal, which I read using the readadc command.



GPS:

To position the robot within the world, as well as in relation to the starting point, there is only really one consumer option: GPS. It is easily jammed and comparatively inaccurate, however, it can work really well. It works by amplifying signals from satellites in geostationary orbit and calculating its position relative to them. As they are stationary relative to a fixed point on Earth, the latitude and longitude.

The module I chose- the Lassen iQ is one of the simplest to interface. It has 8 pins and uses a simple serial interface. The downside is, it has an inset male connector, which means that a very small surface mount header must be soldered to connect it. I managed to surface mount it using school soldering equipment.

Camera:

I also bought a wireless "security" camera, with receiver, which I mounted on top of the robot. This provides a live video and audio feed back to the operator, from the robot's point of view. By buying this as a package, I don't have to combine it with the rest of the electronics- it's a standalone system.

Section 3 – Prototyping—Summary of Choices

Motors

I had the choice between stepper with encoder, DC with encoder and servo. I chose servos because they are strong for their size, contain an inbuilt gear box and encoder and they position accurately. They greatly simplify the control systems.

Power

I decided that for prototyping, demonstrating, testing and developing the spider, using batteries was too inefficient. I modified a computer power supply so that the robot could run from the mains.

Computer Control

Although this will be an add-on later, when I have finished the project, I have concluded that USB is too complicated to attempt without converter chips. I will use a serial interface to communicate with my electronics.

Outputs

Although LCDs and other more complicated screens look nice and are intuitive, they are not necessary for this project. I will use a combination of computers and LEDs to display information

Inter-chip Communication

I2C will allow me to communicate with a huge range of devices using inbuilt commands and without requiring lots of complex programming.

Environment Sensors

A combination of wireless camera and triple axis accelerometer will allow me to sense all I need to about the environment the robot is in.

Navigation

GPS is the only real option available to me, without involving complex maths and processing beyond the limits of necessity and feasibility. I chose a cheap and simple module.

Section 3 – Prototyping—Secondary (Final) Specification

- I will use servo motors to articulate the legs
- I will use a power supply to power the robot, so it must have suitable connectors.
- A serial interface will be added to the controller for computer control.
- The controller will have LEDs and a computer interface to display information.
- I will use the I2C bus for the inter-chip communication
- It will have a wireless camera.
- It will have a 3 axis accelerometer and GPS chip on board.
- It will have solar panels as a back up power system.

These points are in addition to my secondary specification of:

- The robot should be remotely operated.
- It should be capable of sending back information, including a camera feed, data about the surroundings and navigational aids.
- It should be able to cover all terrain, including rough ground such as forests and severe inclines.
- Steps may need to be overcome, so there should be a mechanism to allow this.
- It should have enough power such that it is never left stranded.
- It should be light so that it can't cause any damage to an already devastated area and it can move subtly.
- It must be fast and agile; able to manoeuvre in enclosed spaces.
- There should be a way of manufacturing it in an industrial environment.
- It must have reliable operation.
- The controls must be simple to use but also versatile and powerful.
- The wireless link between device and controller must be reliable.
- It should be made from parts which are easily sourced, in case of damage.
- Water resistivity would be incredibly useful.
- If possible, a microphone output should be also send back.
- If it runs out of power, it must have a backup system to stop it damaging itself or surroundings.
- Reliable and powerful navigational aids should be on board, otherwise it is no use for surveying.
- To assist with the surveying, it must carry equipment to monitor the environment.
- It must be fairly hardy and able to withstand minor damage
- The ground it moves along might well be wet or slippery, and it must overcome this.
- It must not be possible to disorient the moving mechanisms, ie, falling over, rolling etc.

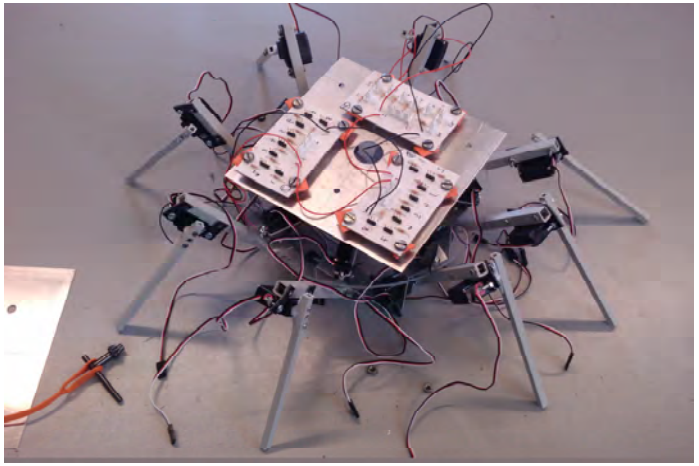
As I had experimented making the chassis and legs before, I knew how to best go about it. I started by working out the diameter of the chassis, to stop the legs hitting each other. I allowed a tolerance, to give enough clearance. By working out a “locus” of where the bottom bracket can move, I could model the spider as an octagon, and work out the diameter of its enclosing circle. However, I chose to do it more practically, and found some discs which were the same diameter as the locus of rotation. I laid these out in a circle and applied my tolerance. The chassis had to be about 21 cm in diameter.

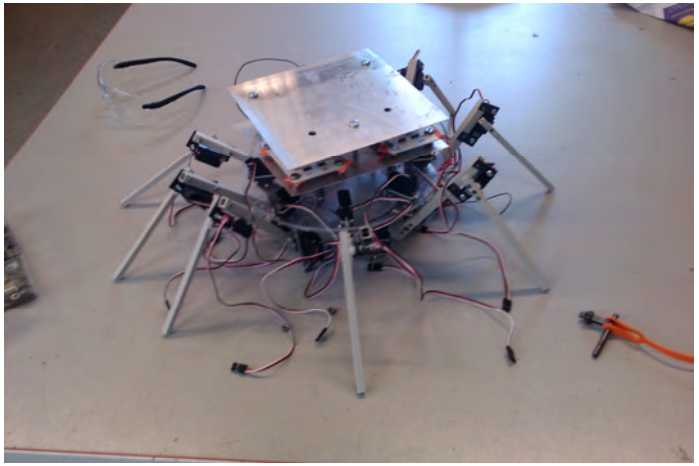
I then selected my material. It had to be lightweight, aesthetically pleasing, and easy to work with. I chose Perspex (acrylic) because it satisfied these requirements. I then went about the same processes as with the prototype chassis, but only drilled a 32mm hole in the centre this time.

Another modification I made was to make the bracket one sided, changing it from a “U” shape, to an “L” shape. This was because the other side was unnecessary, and this saved materials.

The final modification I made since prototyping was to glue rubber feet to the bottom of the legs. These gave a big more grip which was needed on shinier surfaces.

Finally, I slotted the already constructed legs into the square holes and screwed them in.





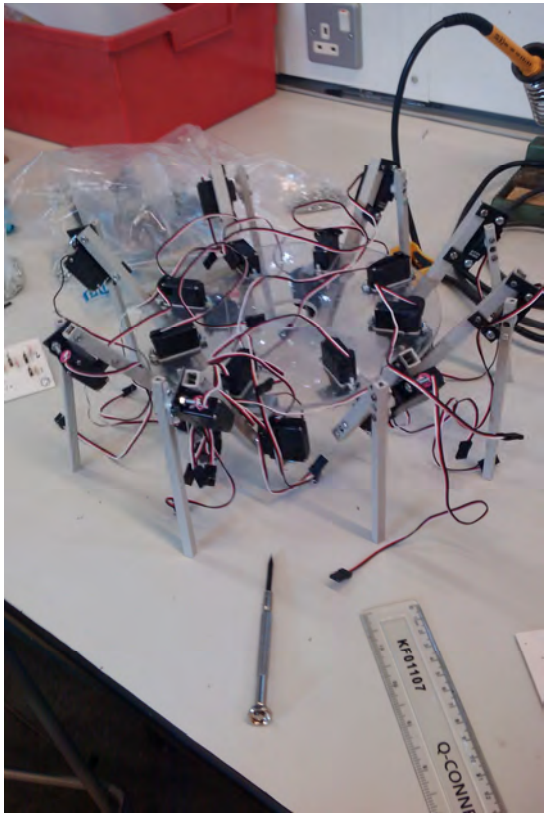
I needed to make a housing for the electronics which would control the legs. Due to space constraints, it had to go on top of the chassis, rather than underneath. This meant that, eventually, a lid would also have to be made.

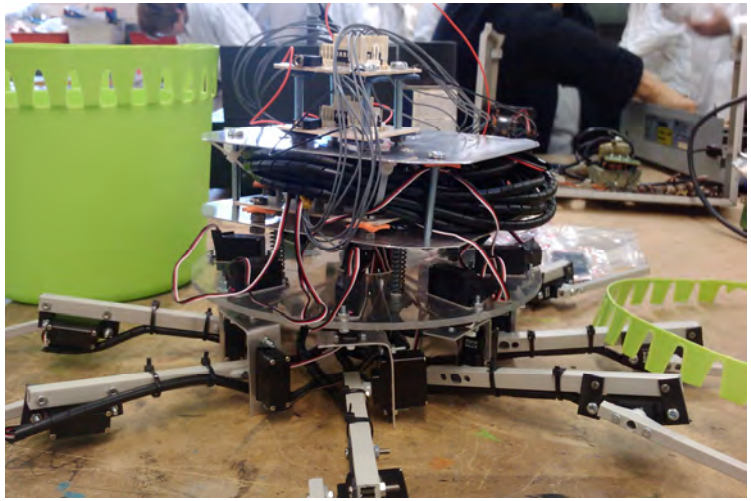
I decided on a platform approach. The boards (In the Original Electronics section) would be arranged so that the ones in direct communication with the legs would be lowest (to reduce wiring), and the radios and external communication circuits would be highest (to reduce interference).

To protect the electronics, I wanted to add a form of suspension to the platforms. This would absorb the shocks of sudden movement and would also prevent the fixings of the platform being sheared.

I decided to use aluminium for the platforms, because it is light, strong, and easy to work with. It is also a bad conductor of electricity, which could reduce the damage of a short.

Starting with square platforms, I drilled central holes for wires, bolted them together and onto the robot, adding springs on the bolts and bolting the circuits to the platforms. I then rounded off the edges using tin snips so that they didn't protrude past the limits of the chassis.





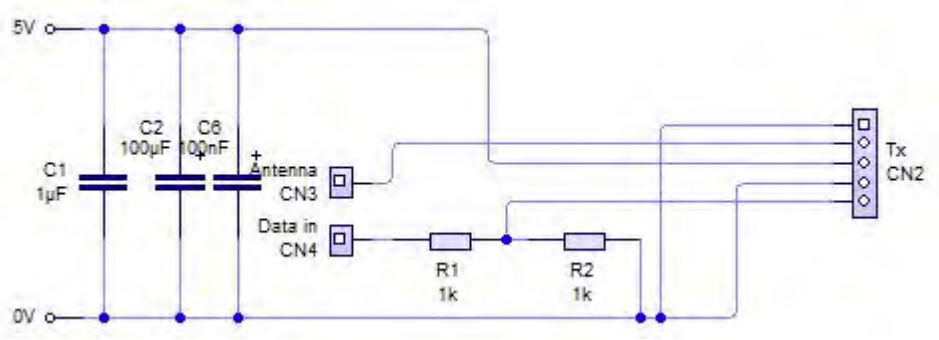
The wiring between platforms was very minimal, which was the aim of my arrangement.

The electronics needed to be enclosed to prevent damage, build up of dust and to help waterproofing. I started off with a green plastic bin, which I cut down to size and sprayed black. The inside being green and the outside black looked nice, but the bin was far too tall for the robot. So I had to rethink.

To continue the Perspex theme, I sourced some 220mm diameter Perspex pipe. This would protect the robot nicely as well as allow it to be see through, which I liked.

To go on top, I made another Perspex disc, drilled holes for the antennae and attached Velcro pads for the solar panels. I then glued this to the top of the tube.

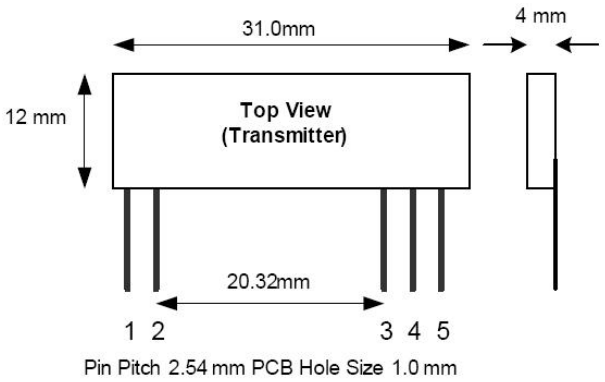
Section 4 – Making - Radio Modules

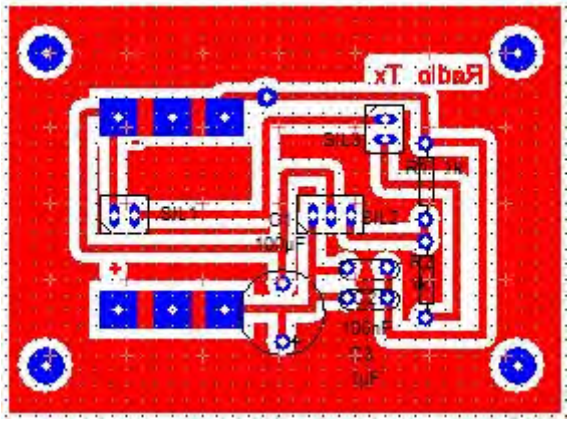


Tx

I chose Easy radio modules as the wireless link between the controller circuit and the spider because they have a high quality signal due to the fact that all of the signal processing and conditioning is handled by default. This means, once connected correctly, the input of the transmitter to the output of the receiver behaves exactly like a wire. They have two major disadvantages however, they are relatively more expensive and the default baud rate is 19200, which can only be modified by passing a signal of this high speed into the module.

The transmitter was the easier of the two modules to connect, requiring a header for the antenna, an input (connected to a PIC), smoothing and power. There is a potential divider on the input pin because the module is powered by 5V but requires a ~3V signal. The potential divider taps off the extra voltage to protect the pin.

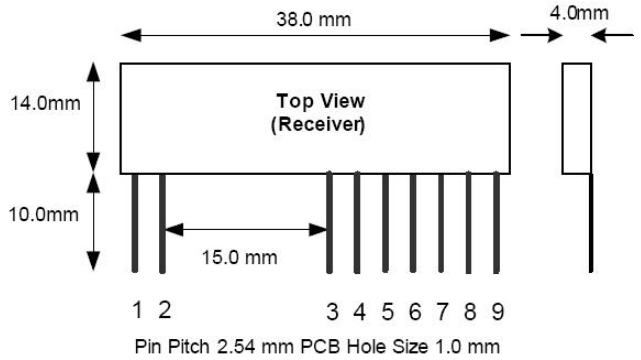




The physical module has uneven pin spacing (as shown in the diagram). To attach it to the PCB, I used a 2 pin header and a 3 pin header, correctly spaced. In the PCB diagram, these are SIL1 and SIL2.

To help reduce interference and to dampen the noise from the circuit, I added a ground plane and smoothing capacitors physically close to the module. They had to be close, because small ripples are localised.

Pin No	Name	Description	Notes
1	RF Gnd	RF ground. Connect to antenna ground (coaxial cable screen braid) and local ground plane. Internally connected to Pin 4	
2	RF Out	50Ω RF output. Connect to suitable antenna	
3	Vcc	Positive supply pin. +2.5 to +5.5 Volts. This should be a 'clean' noise free supply with less than 25mV of ripple	
4	Gnd	Supply 0 Volt and Ground Plane	
5	TXD	Transmit Data Digital Input (SDI)	



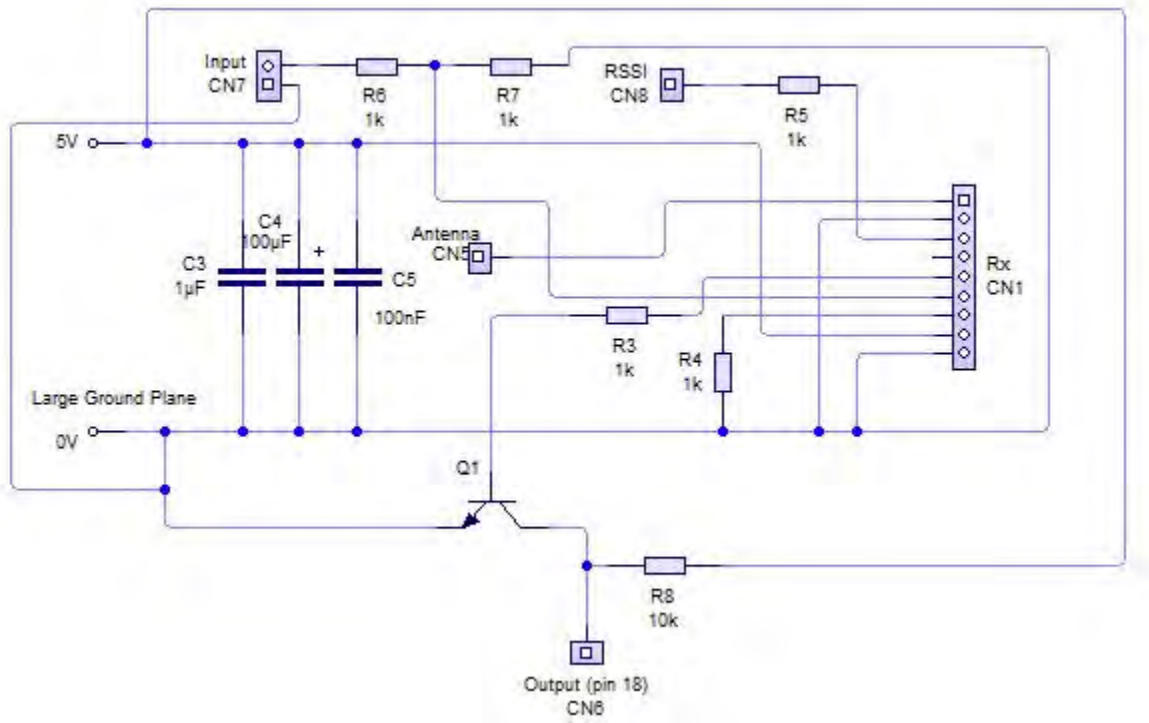
Rx

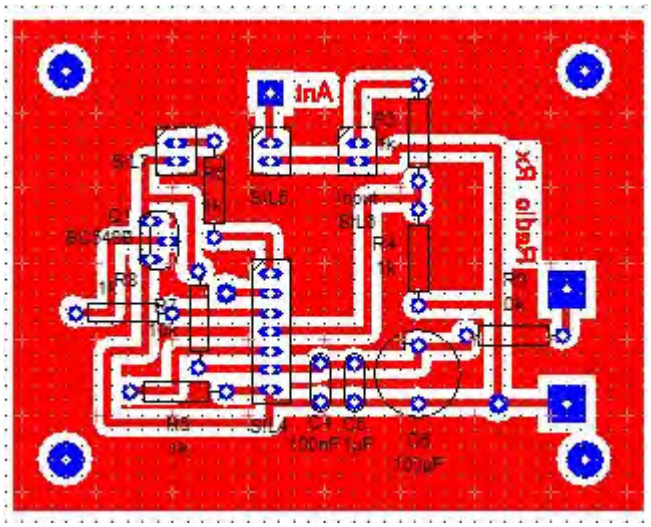
The receiver was much more complicated by comparison. It required a similar potential divider on the input (for the same voltage reasons), a header for the antenna and smooth 5V power lines.

Because the modules invert the data by default, and I wanted the modules to act in the same way as a wire, Q1 inverts the inverted signal back to the true data. R8 is an input pull down and R3 is a protective resistor.

RSSI is an analogue indication of the strength of the signal. I added a 1K protective resistor onto this, to protect the input it will lead to in case anything goes wrong.

The last component to note is R4, the 1K pull down on pin 7 (RDY). This pin must be tied low for the module to receive data. It acts as a flow control which is unnecessary in this circuit.





Pin No	Name	Description	Notes
1	Antenna	50Ω RF input/output. Connect to suitable antenna.	
2	RF Ground	RF ground. Connect to antenna ground (coaxial cable screen braid) and local ground plane. Internally connected to other Ground pins.	
3	RSSI	Received Signal Strength Indication - Analogue	
4	BSY	Output (Low - Ready for data from Host) (High - Not Ready)	CTS function
5	Data Out	Received Data Output	SDO
6	Data In	ER command Input	SDI
7	RDY	Input (Low – Host Ready to receive data) (High – Not Ready)	RTS function
8	Vcc	Positive supply pin. +2.5 to +5.5 Volts. This should be a 'clean' noise free supply with less than 25mV of ripple.	
9	Ground	Connect to supply 0 Volt and ground plane	

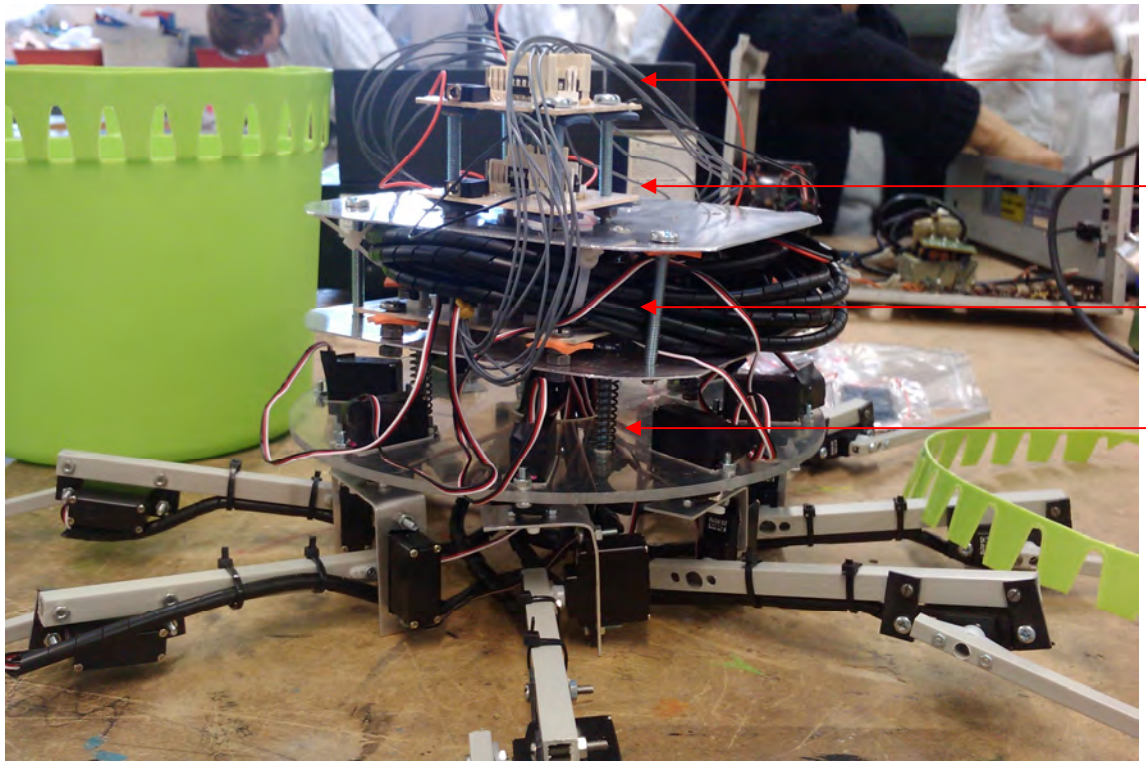
Section 4 – Making - Original Electronics

I took a modular approach in the electronics for the robot so that I could change elements of it if I needed to. It also allowed for better testing and modification.

I therefore created “power boards” which power the servos and link all of the data lines to a central header, through a unique 330 Ohm resistor for each line. This allowed me to reduce the wiring on the main boards by 2/3s and keep the servo power separate from the 4.5V chip supply.

I placed this set of boards at the very bottom of the stack, close to the servos.

On top of those are two identical boards (with different names) which power the chips and have headers on the outputs to connect directly to the headers on the power boards. Above this (to increase range) is the radio receiver board, and above this is a PCB with a copper strip to attach to the aerial.



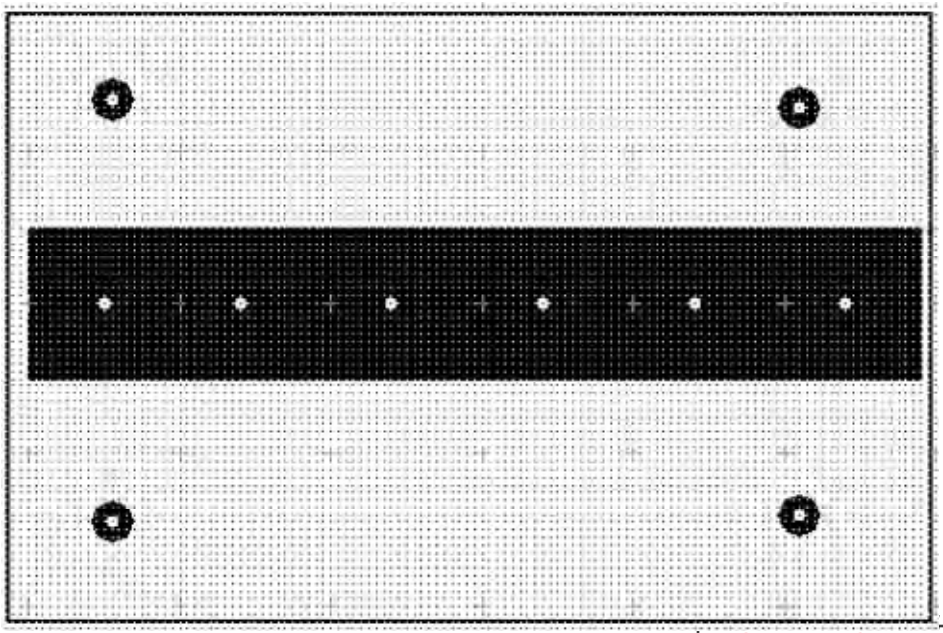
Main PIC board

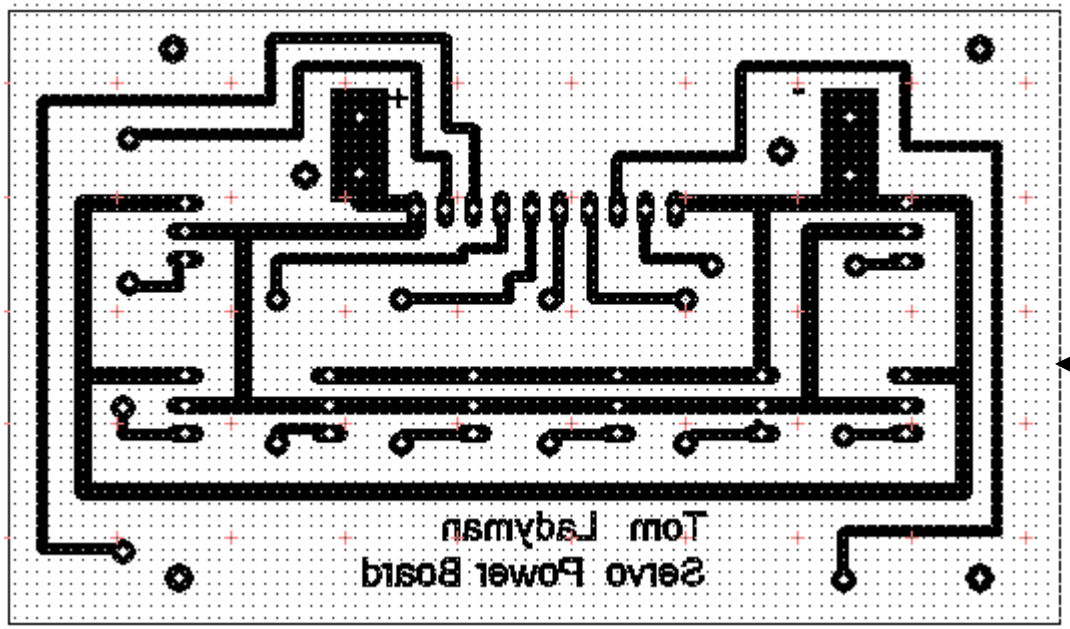
Slave PIC board

3 power boards

Suspension pillar (bolt and spring). This absorbs shock and reduces the risk of damage to the structure.

The radio and antenna would be mounted above this stack.

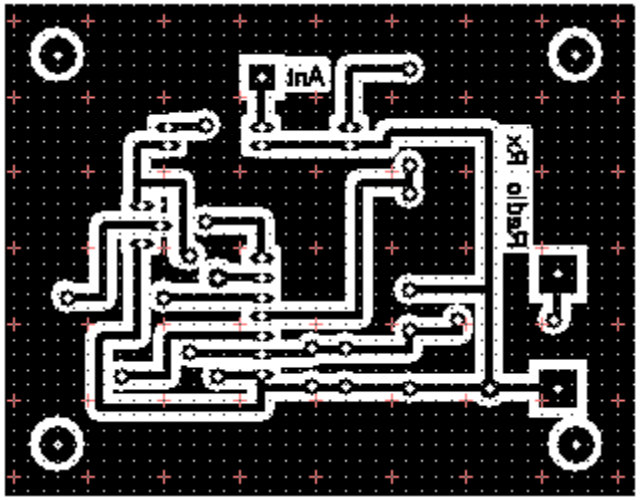


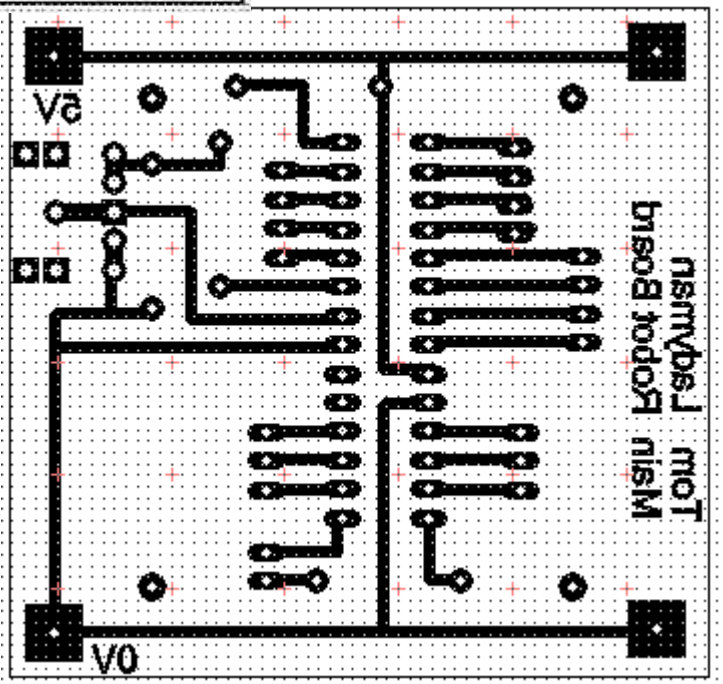


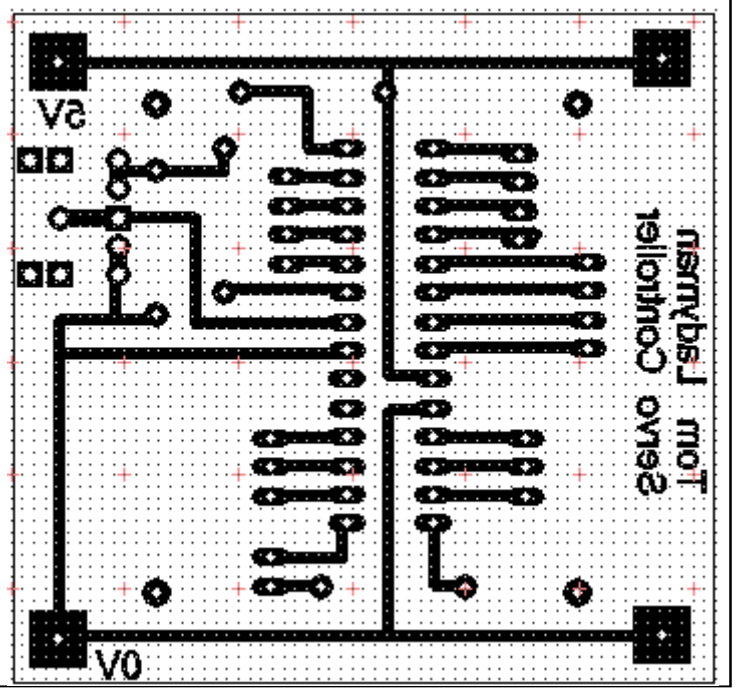
Power board (x3)

Aerial board (above)
Radio receiver (below)

Main board (right)
Slave board (far right)

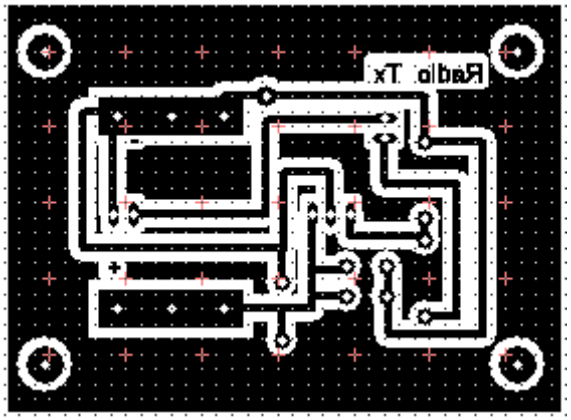






Section 4 – Making - Original Electronics - Off Board + Testing the Servos

Radio:

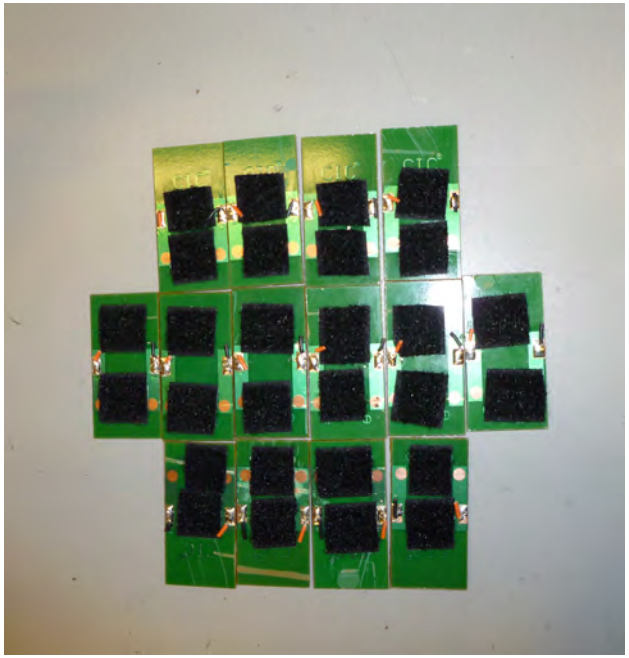


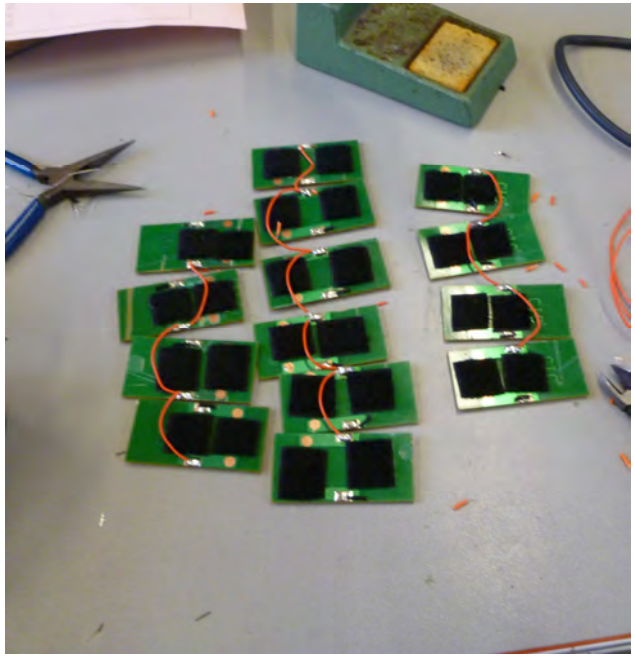
It was impossible to test using all of the servos, due to the fact that my circuit boards were not made to carry the peak of 24 Amps. I tested them with as many servos as I could, and got a few movements working, however I quickly realised that these boards were not the final solution. They are very versatile however, and I re-used them for testing purposes later in the project.

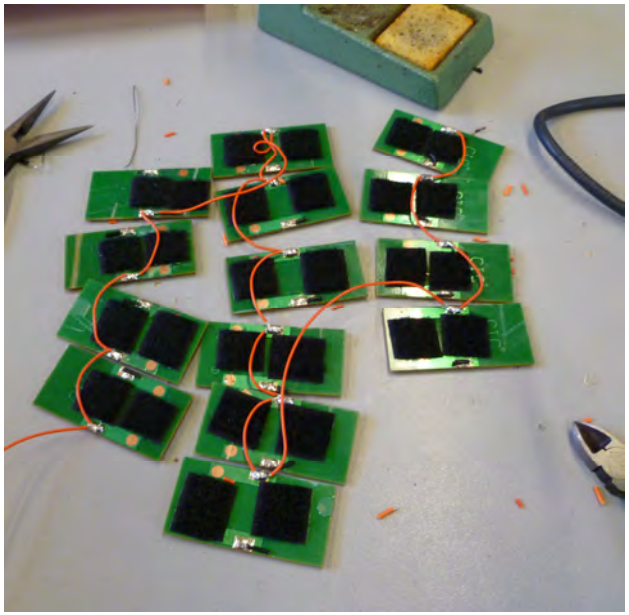
Key Matrix:

The controller would need more input switches that the inputs available. I decided to make a key matrix to scan through the switches and see if they're on. A matrix of switches uses outputs on each column and inputs on each row (or the other way round). Each column is turned on in turn, and the the inputs are checked in turn. This allowed whichever switch is on to be identified.

Section 4 – Solar Panels







This was the only element of the original element which I kept in its entirety.

To ensure that the robot never gets stuck out of power and is unable to move back, I decided to fit a back up power system. This will power the navigation equipment, camera and radios, making it unlikely that the robot will be irretrievable.

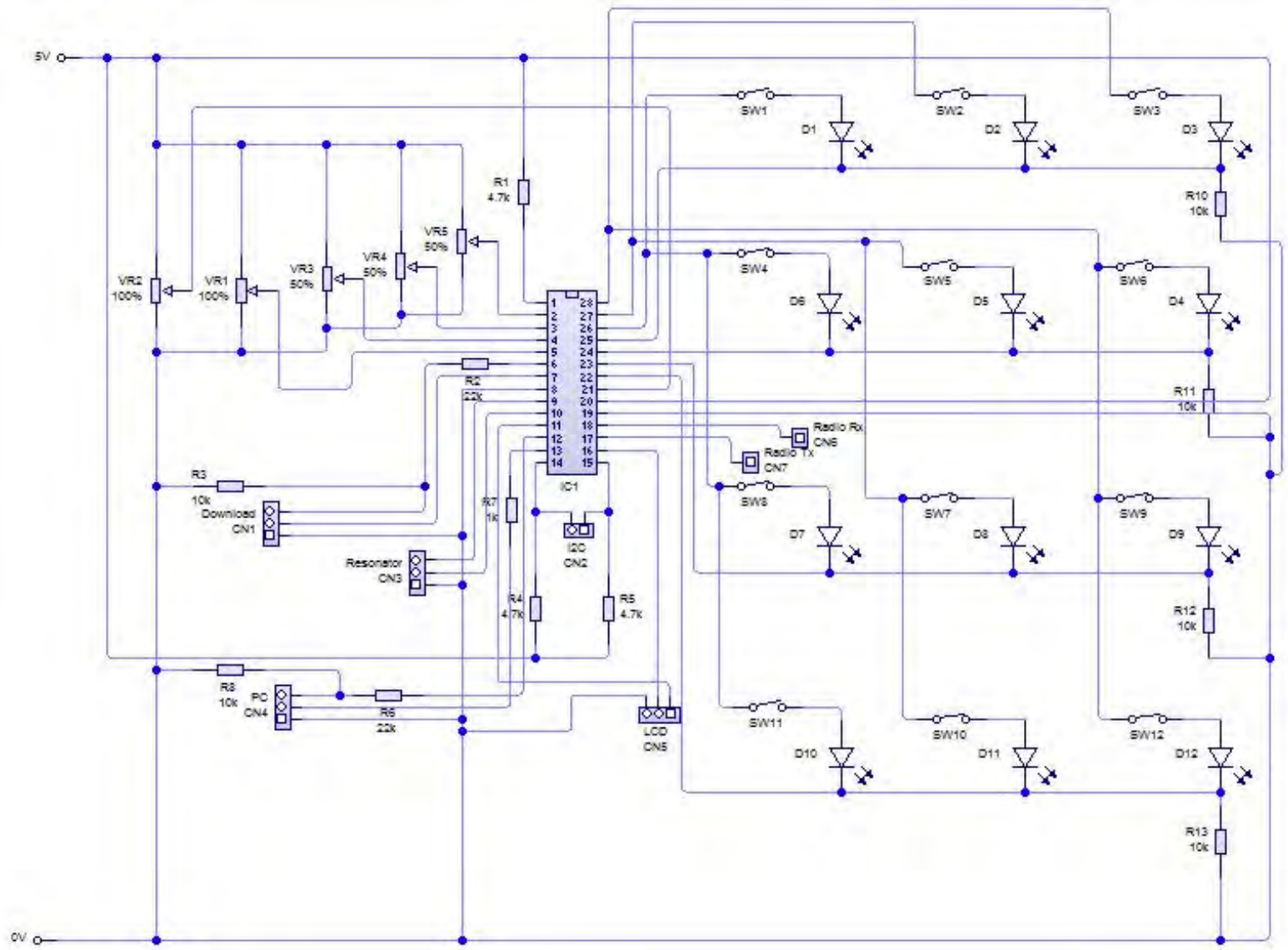
I put 14 solar panels onto the lid of the robot. I chose 14 because they fit into a nice shape, and produce 14V, which is ideal for trickle charging a 9V battery- the storage of the backup power.

To fix them on, I used 6 rows of Velcro. This will allow me to move them around or replace damaged panels easily. It is also very strong and doesn't damage the panels like making screw holes would.

Section 4 – Making - Final Systems Diagram and Circuits

Controller schematic:

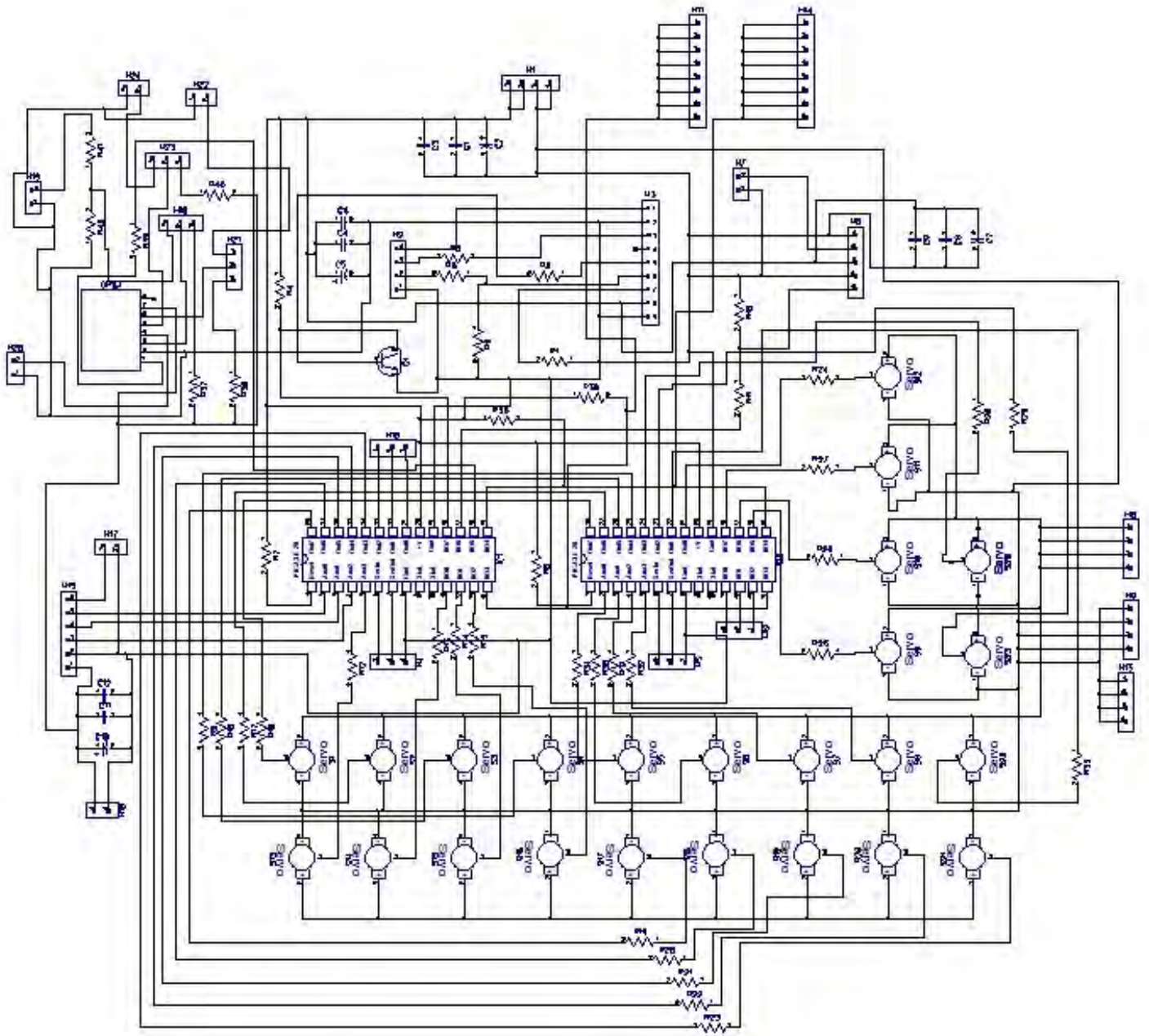
The headers “Radio Rx” and “Radio Tx” connect to the transmitter and receiver radio circuits, respectively.



Section 4 – Making - Final Systems Diagram and Circuits

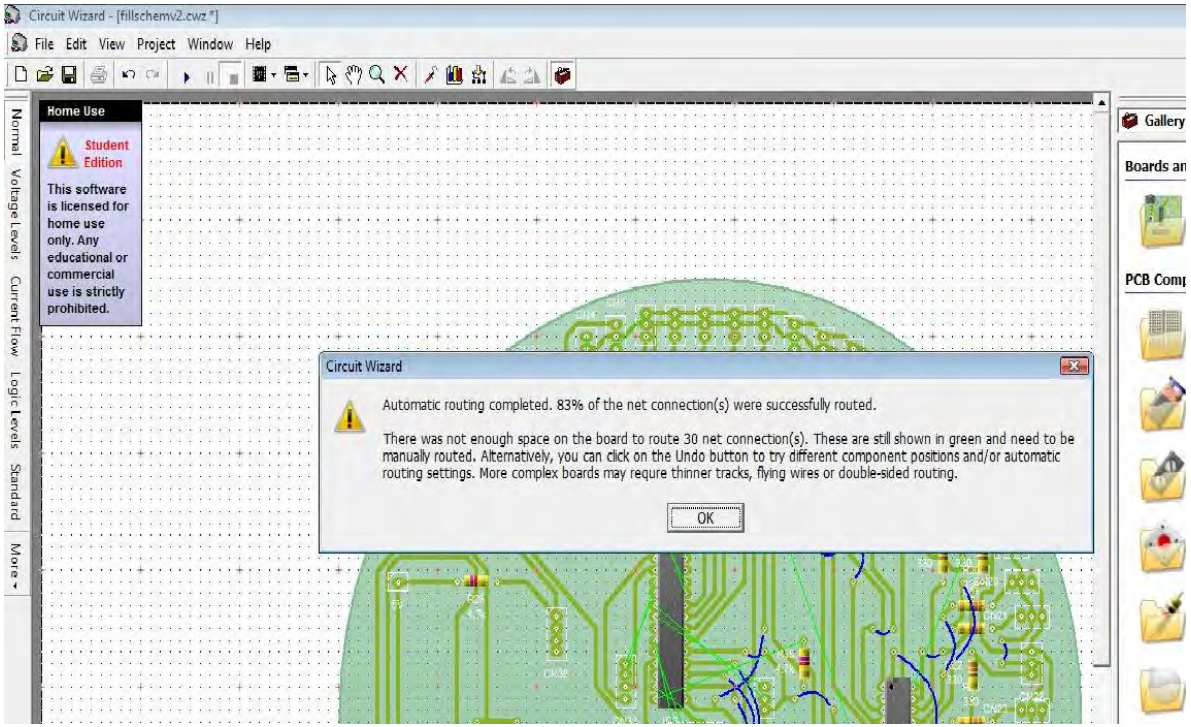
My original design for the electronics was not robust enough– it didn't allow for many of the more advanced features, such as the GPS, it was also very heavy and space inefficient.

This is the design I built the final board from:

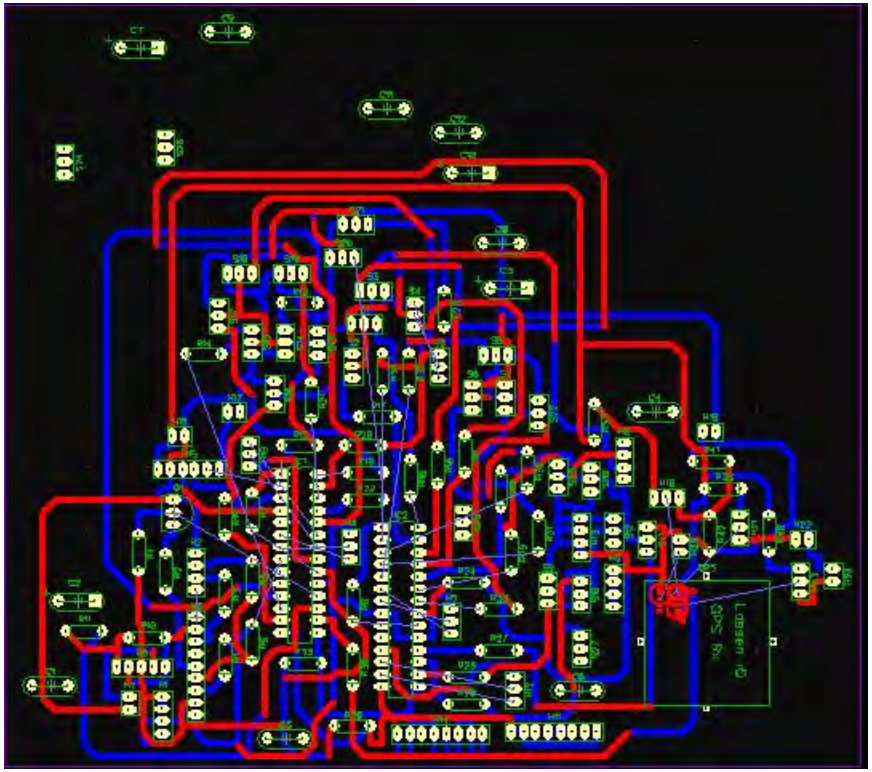


Section 4 – Making - Board Design

I discovered that the most important element in the design of the spider was weight. I realised that having two platforms and main circuit boards, springs and bolts was not weight effective and worked out a better solution. By putting all of the components onto one, large circuit board, I could alleviate the need to use platforms. This would also decrease the height of the robot. Furthermore, by making it octagonal, I could vastly reduce the amount of wiring, because each leg would have a side of the octagon dedicated to it. I attempted an initial autoroute using Circuit Wizard, the software my school use.



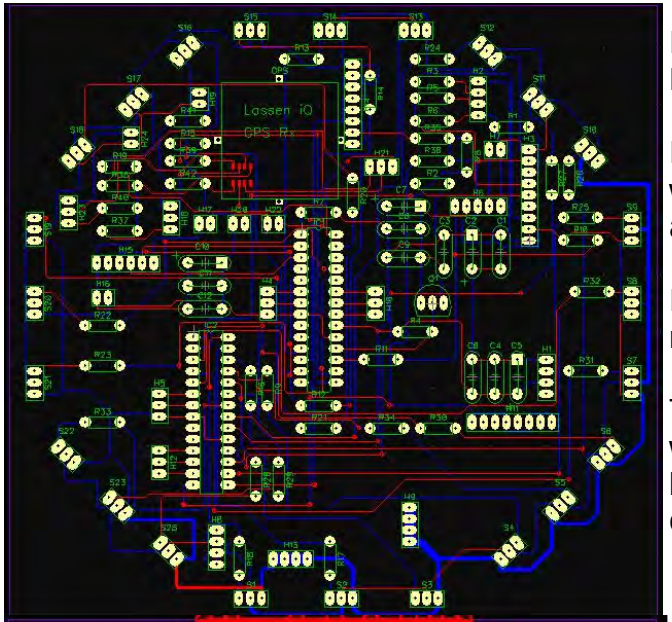
As you can see, only 83% of the connections could be routed, and a lot of link wires were used. Circuit wizard is also incapable of making more complex shapes. I looked into other software, such as Eagle, and settled with Diptrace.

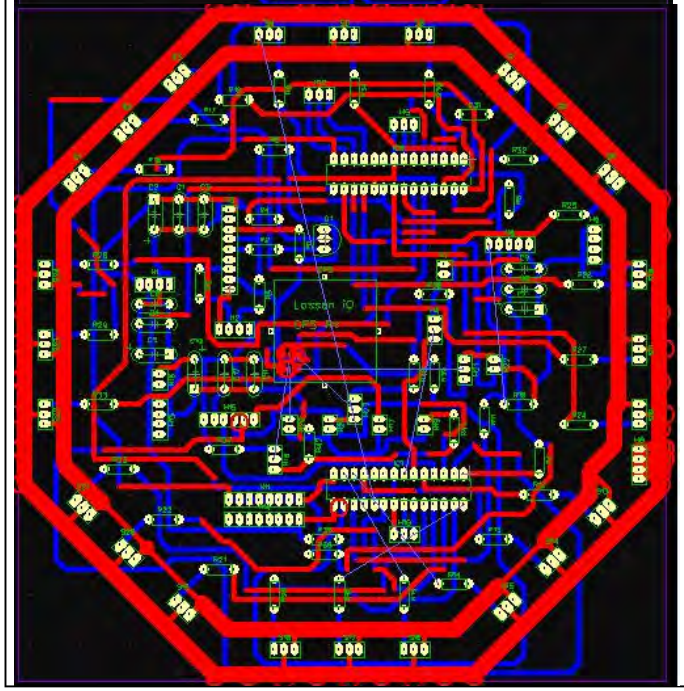


I emailed Novarm, the company who make it and asked if they would agree to give me a free product key, and they did. This allowed me to use other features not available in the free version, but, more importantly for me, use a lot more components.

The photo on the left shows the first autoroute I made in Diptrace. It has a much higher success rate than circuit wizard, and didn't use link wires. The purple lines are the unconnected pins. Blue is the bottom copper layer, red is the top copper layer, white is the pads and green is the top silk screen.

The best thing about drawing the PCBs with Diptrace is that it marks which pins should be connected with purple lines. This means that, with an accurate schematic, it is impossible to make a mistake.





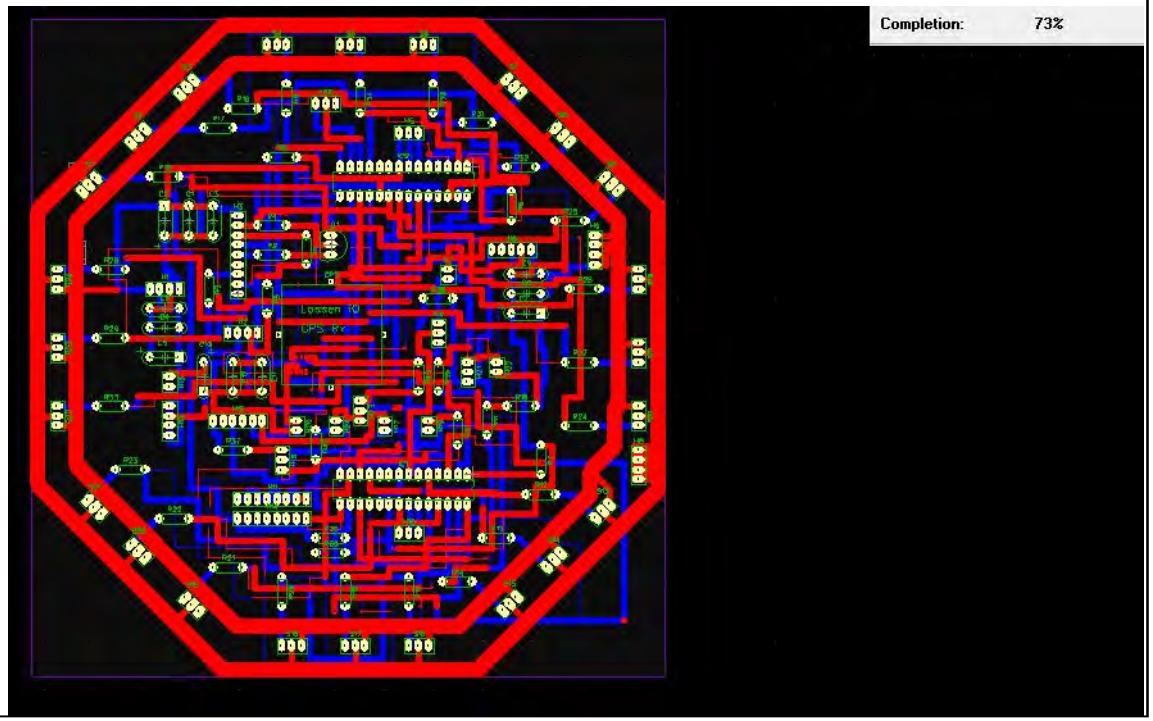
I arranged the components in an octagon and part auto routed and part manual routed to get an initial PCB design (shown left).

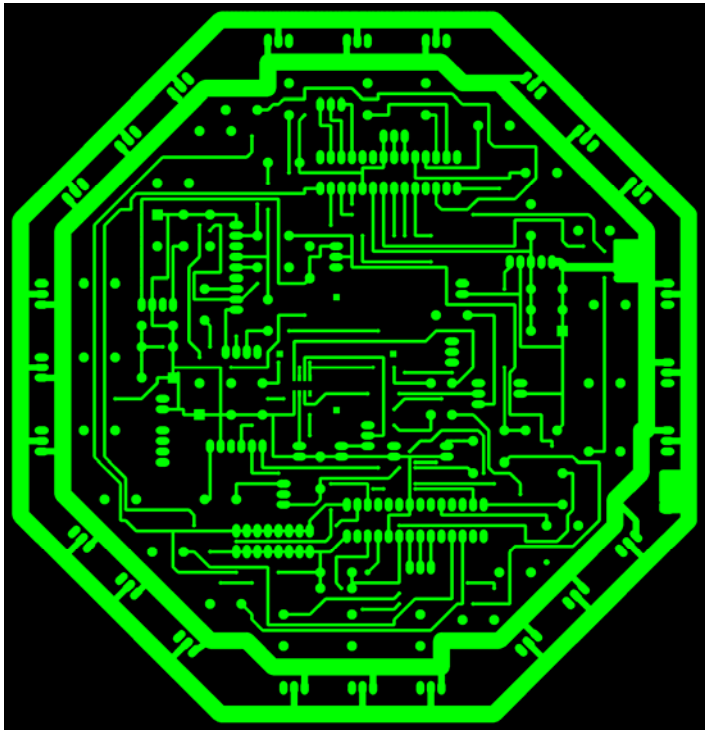
It would function fully, except the power rails for the servos are much too thin. If powered up, the board would burn itself out from the sheer current. I was also unhappy with the messy square connections around the edges and the different sized tracks.

I decided to redo it with the 6V servo power rail on the outside, in an octagon shape. Parallel to this rail, I ran the 0V rail, which was common. This allowed easy access for all of the components.

Trying auto routing one more time, I produced the result in the bottom left (bottom right shows the software working). For some reason, it chose to add extra corners onto the sides, however, produced much better results than before, even with thicker tracks by default. It still, however, did not make all of the connections.

This new octagon shape was certainly the correct path though.



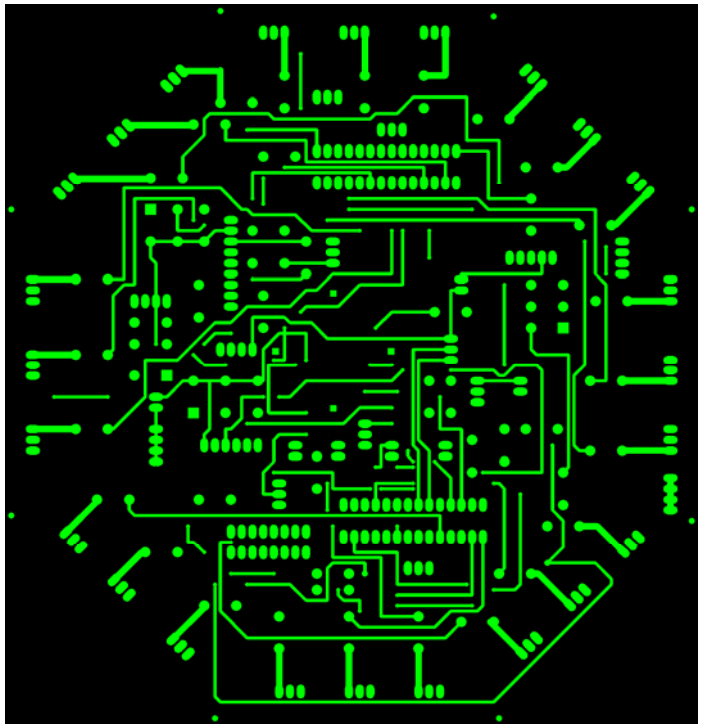


Top Copper

I realised that the best way of doing it to my standards was to manually route every connection, as I have done here.

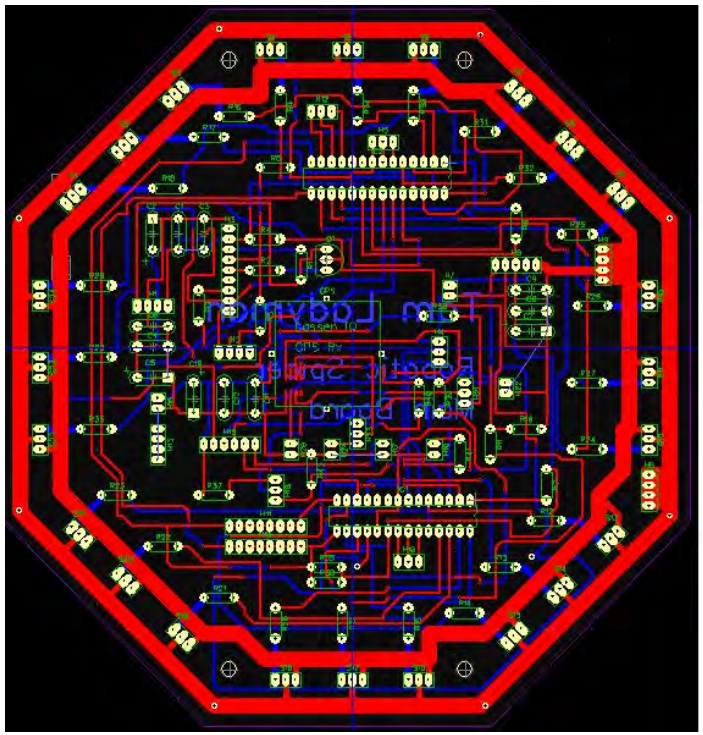
This was the design I sent to be made.

Four mounting holes were also added, and my name and project title on the bottom silk screen.



Bottom Copper

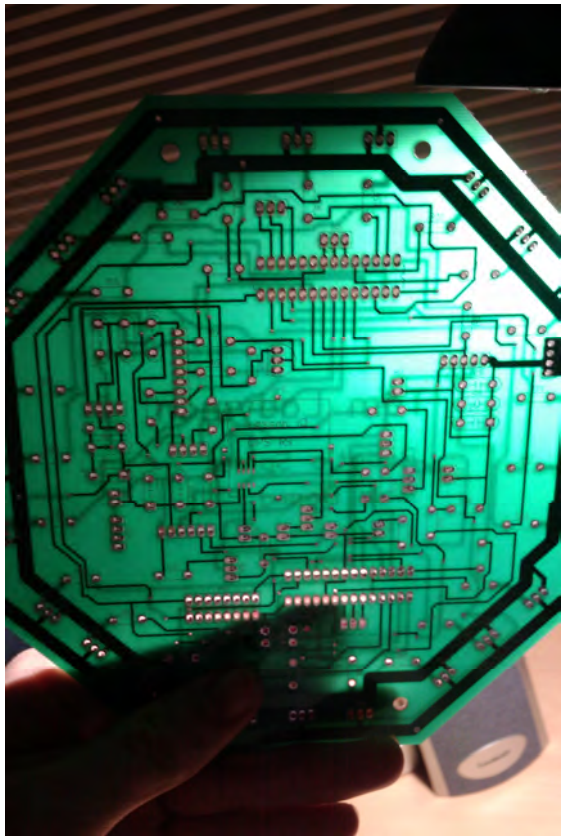
Full View

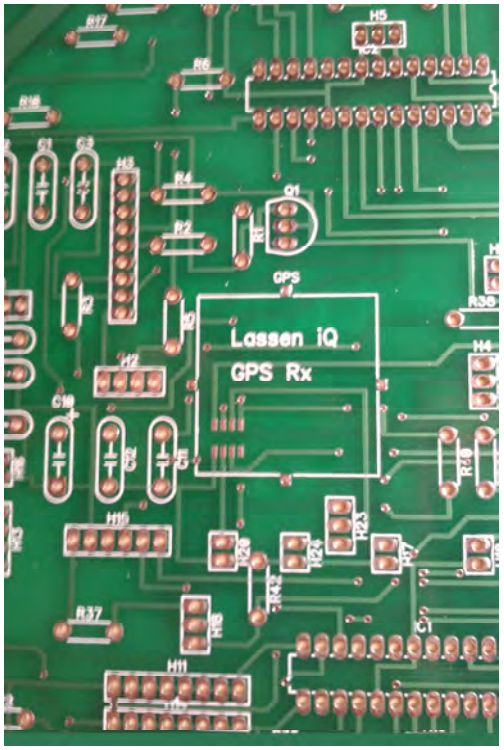


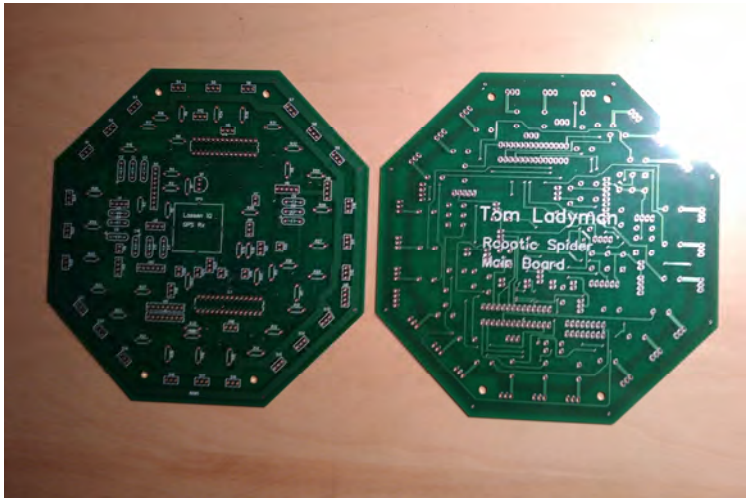
I sent the board to be made, and two copies arrived, which allowed me to have a spare.

Before I used it for the first time, I checked the connections, and there were a tracks which had passed the software checks, but shouldn't have done so. These included a couple of unconnected resistors and some crossed tracks. The most major was that the SCL and SDA lines of the I2C bus were shorting with each other.

I fixed these problems by breaking tracks and scratching away solder mask with a knife, and adding the required link wires.

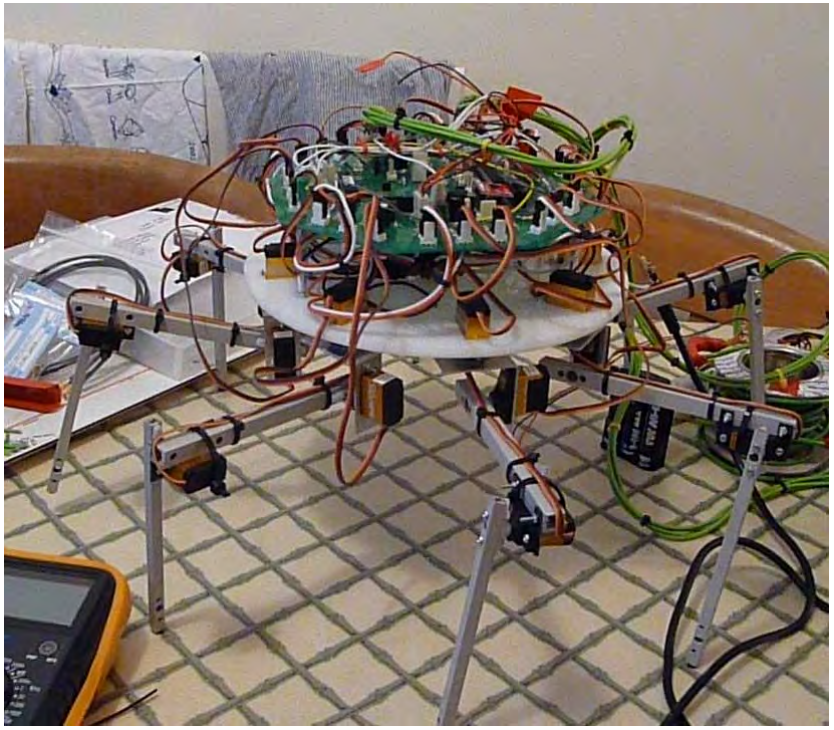




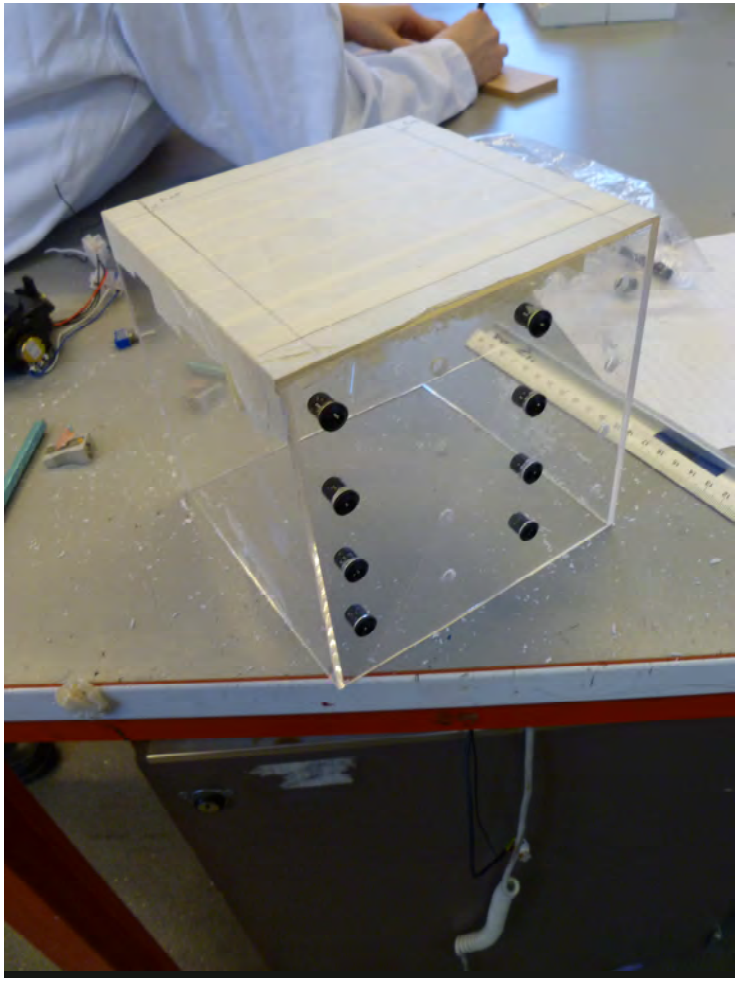


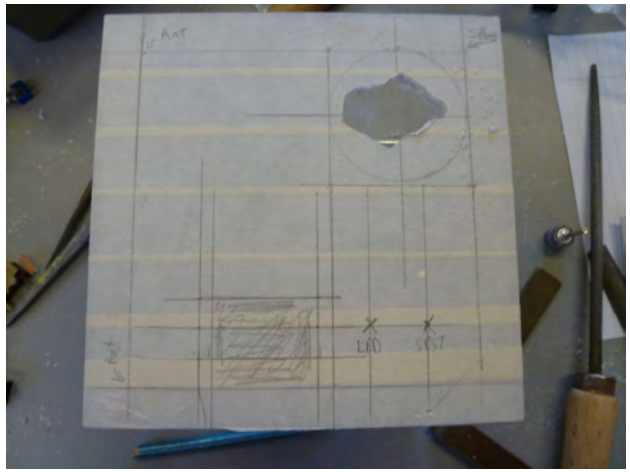
The board was incredibly light, and very well made. It took a huge amount of solder to populate, and was substantially heavier afterwards.

There was one surface mount component, which I "painted" solder onto using a standard soldering iron and a standard tip. It took two attempts to get it right, as using slightly too much solder shorted out all the pins and was impossible to fix.



To locate the new board, I used some metal rod to make 4 “studs”, which are double ended bolts, and 4 acrylic spacers to support the board. I used a lathe to cut the acrylic tube down to size and round it off. I then cut the metal rods down to the right size and used a tap and die to put the thread on the end. I was careful to use enough oil and keep the die as flat as possible. This ensured a straight thread.



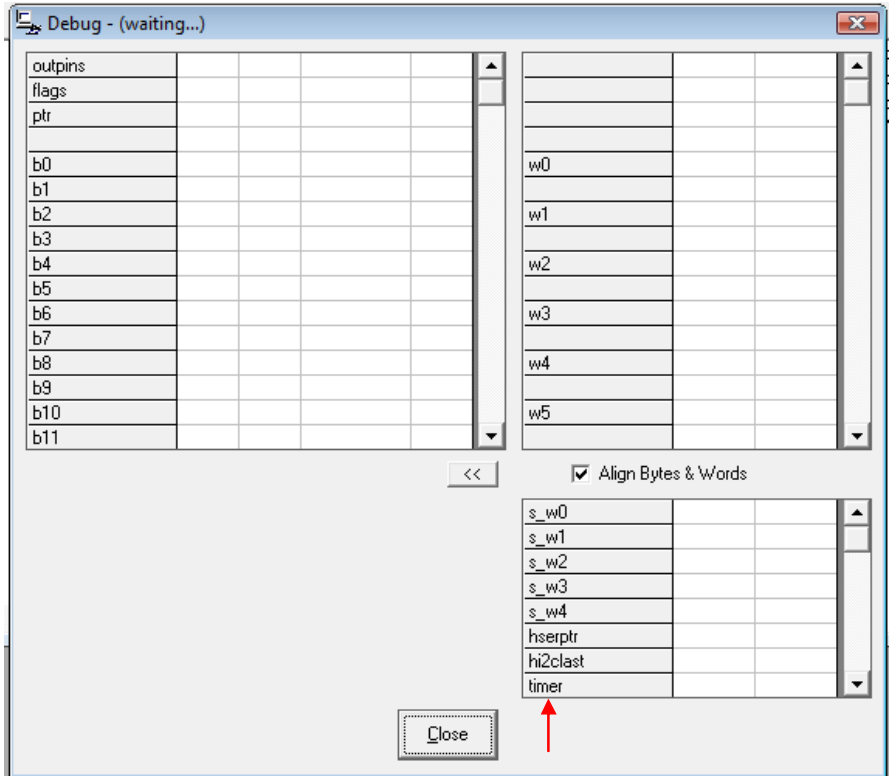


I wanted a stylish and functional controller, so I decided to stick with the Perspex theme from the robot. After ordering a Perspex cube from the internet, I made a base to fit inside and painted it white. This was big enough for the key matrix board to fit on top of. By taping over the sides, I was able to accurately mark out the Perspex surface, and by using a fast Dremel drill, conical reamer and by taping the inside for extra strength, I drilled out all of the holes and spaces without it shattering. The larger holes, such as for the joystick, were made by using a sanding belt attachment to the Dremel.

The holes on the side are for switches and LED indicators for the switches and there is an off switch near the bottom of a face.

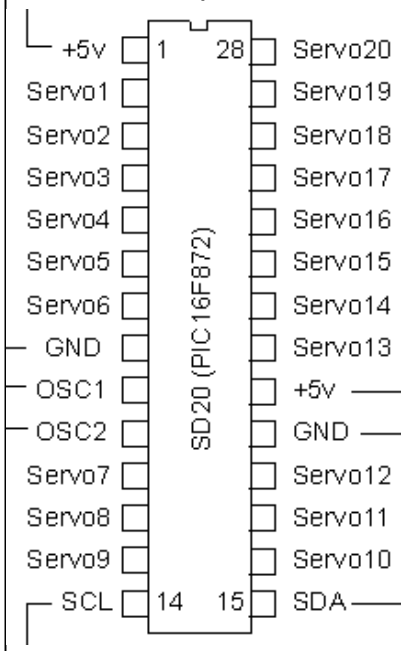
While testing the new board, I came across a couple of problems.

Although the PICAxe 28X1 has many pins to use as outputs (and two of them have far more than the necessary 24), the servo command uses an internal register as a timer. This register is simply called “timer” (as shown in the debug screen) and is the hardware timer for all of the processes on the chip. This includes Hi2c, servo and hserin.



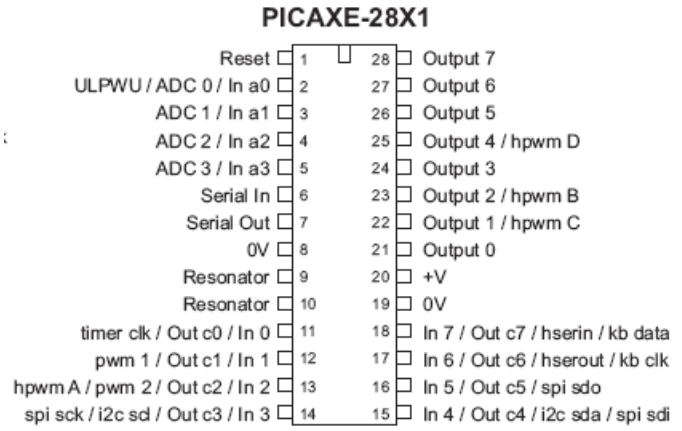
This caused a major setback as all the processes were stalled by the servo command using the register, and the servo command was stalled by the other processes, resulting in limp servos. I researched other methods of controlling servos, such as manually creating the waves myself. That way, I could specify which registers to use as timers. This was far too complicated and, itself, had flaws, ie the PICAxe bootloader slows down the actual speed of the chip, so carefully calculated timing sequences are useless.

The other problem was that the board was solder masked and double sided, this meant that it was impossible to make major alterations to the circuitry. One board had all of the circuits on it too, which meant that I couldn’t feasibly restart that section.



I considered building a daughter board to slot into the chip holder of the slave 28X1 and build a servo driving circuit on top of that, and then I considered wiping the PICAxe bootloader off the chip and programming it in C to make it a server driving chip.

However, I remembered the SD20 chip that I had researched earlier in the project and compared the two pin outs: they were almost identical. I had to put a resistor onto the open drain output of the SD20, solder in a resonator, transfer the servo wires to it and change the reset resistor to a link wire. Because I wanted to free up the PICAxe timer, I migrated all of the servos across, even though it meant connecting the hinge joints of opposite legs, to give a total of 20 servo channels.



Section 4 – Making - Programming

A note on programming style.

I used a 28X1 as the onboard controller for the spider. This is neither the fastest, nor does it have the best storage capabilities. It is however, very forgiving when mistakes are made with connections, and has all of the major functionality of the more expensive chips (such as the 28X2).

Because of this downgrade in technology, I had very few variables to work with, and not as much storage space as I would have liked to have. This is why some of the code is presented in a slightly non conventional way, such as the offsets (to save variable space) and the update subroutine (to save space).

Although these areas are non conventional, I believe they have given the best performance possible, probably exceeding the performance of a badly written, but conventional, program written on a higher level chip.

Where possible, I have annotated the code, but it is definitely best to refer to the leg diagrams (later on in the folder) if trying to follow the code, as these show a visual representation of the movement that is very hard to gauge from code alone.

For the purposes of communication, clarity of code and clarity of thought, I have numbered the legs, and labelled each axis of movement. Leg one is the north facing leg if the writing on the board is from left to right. Subsequent legs are labelled clockwise from there. The servo directly attached to the base controls axis A, axis B is the servo mounted to the bracket, and the hinge joint is axis C.

A slightly confusing but important point to note is that PICAxe uses variables with the letter b in front of them. For this reason, the program refers to axis B as axis D. The rest of the labels are as expected.

Also, I have assumed 127 to be the centre point for each servo, which explains why this number is so often repeated. I wanted the robot to stand in a “rest position” with all of the servos centred (position 127).

To assist in programming, I created a form of diagram to represent the spider's leg movements in a 2 dimensional plane. The diagram on the next page explains how the diagrams work, and the following pattern is the first one I wrote, detailing how the whole spider can stand up from a reset- where the legs could be in any position. Then there is a pattern to rotate the spider on the spot.

After this movement, there is my first attempt at a walk pattern. I found that it was far too clumsy and remade a walk pattern, which is directly after the initial one.

I then wrote subroutines to fulfil these movements. The subroutines can be called from anywhere in the program, making it a nice modular approach to the program.

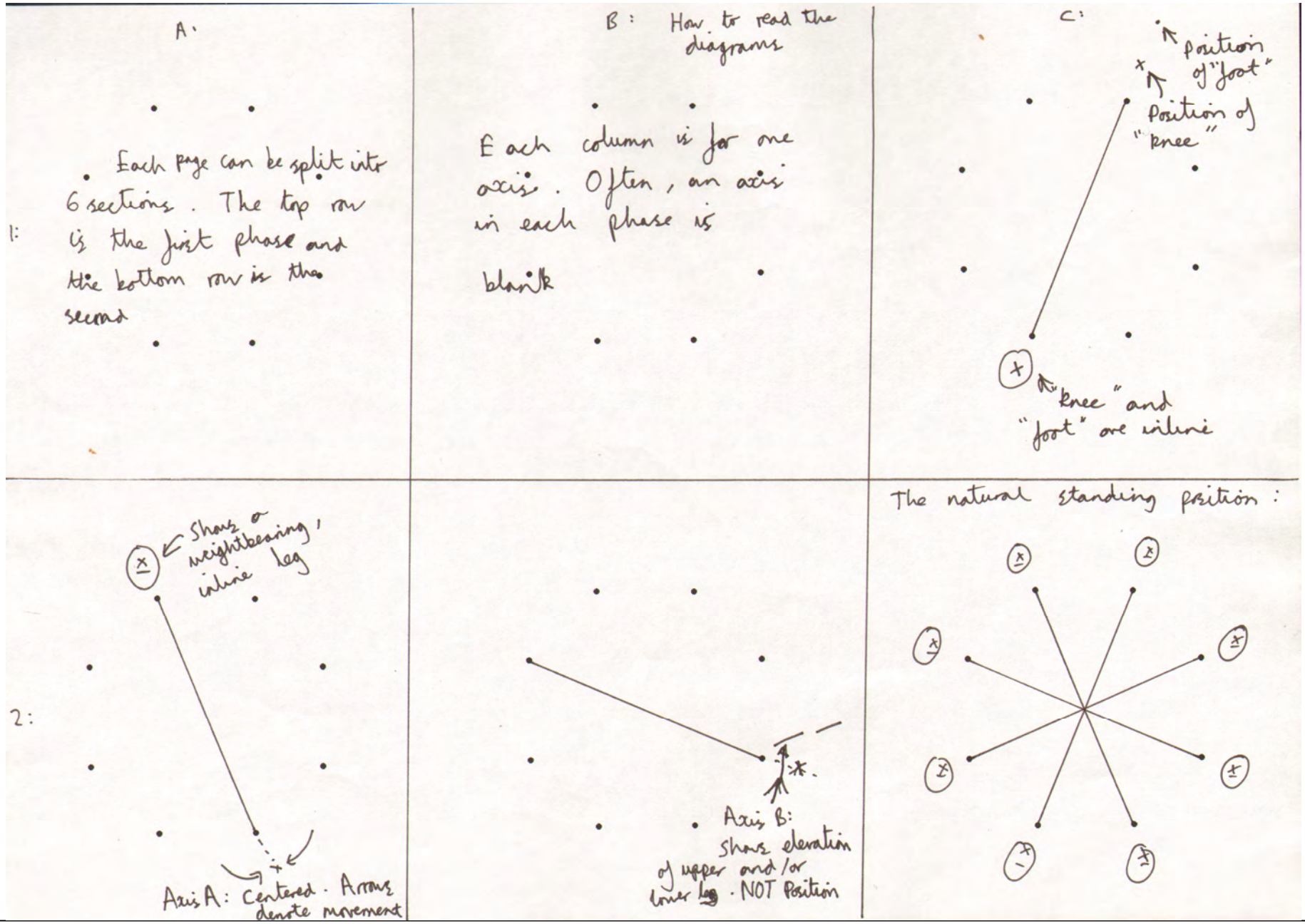
These programs can be found after the diagrams.

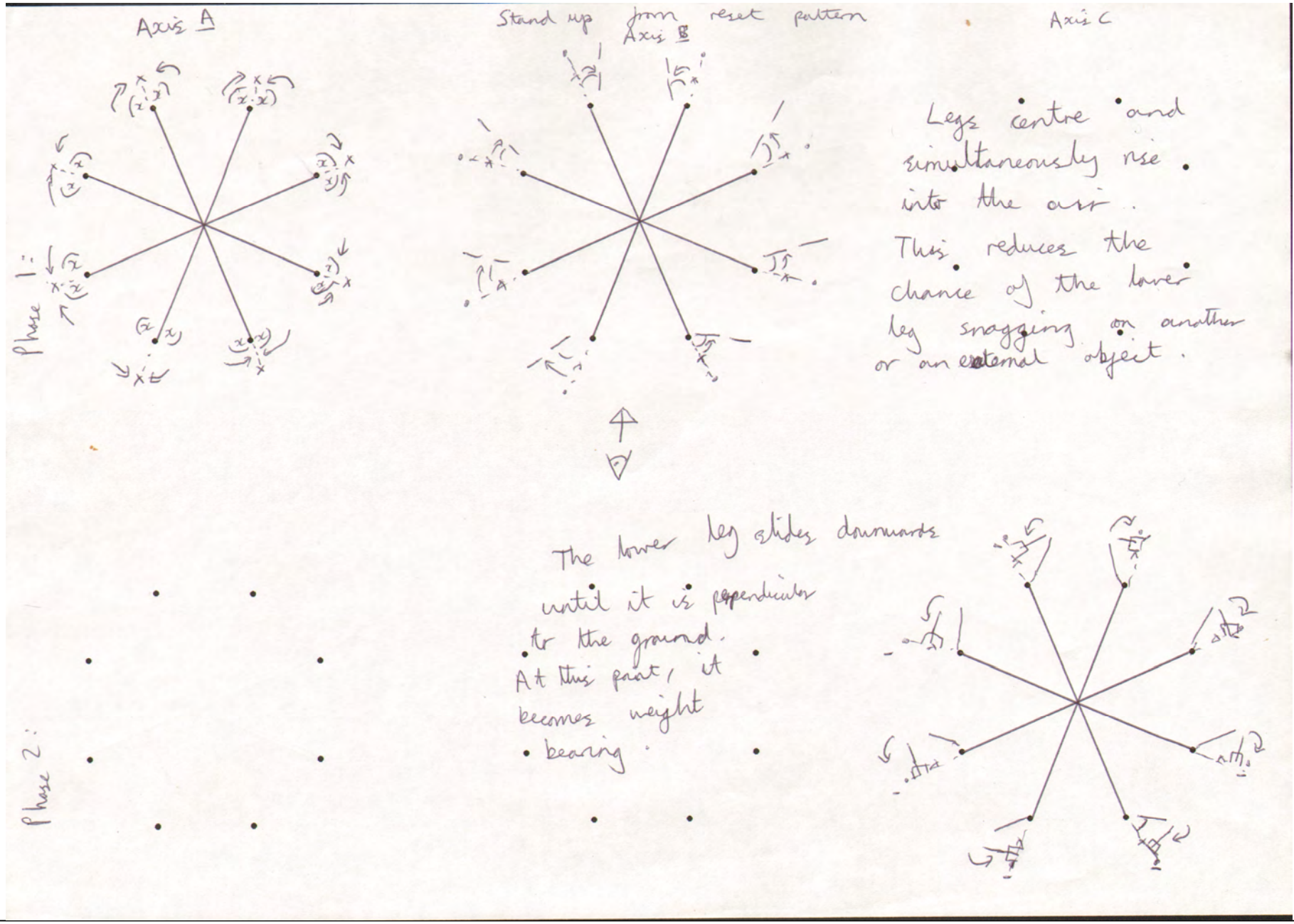
This is a test code to show how the update subroutine was written- it is particularly important in relation to the rest of the code.

```
'
'Program for main robot board
'
init:
#no_table
#no_data

'Start i2c
HI2CSETUP I2CMASTER, $C2, i2cfast, i2cbyte

symbol a1 = b0
symbol a2 = b1
symbol a3 = b2
symbol a4 = b3
symbol a5 = b4
symbol a6 = b5
symbol a7 = b6
symbol a8 = b7
symbol d1 = b8
symbol d2 = b9
symbol d3 = b10
symbol d4 = b11
symbol d5 = b12
symbol d6 = b13
symbol d7 = b14
symbol d8 = b15
symbol c1 = b16
symbol c2 = b17
symbol c3 = b18
symbol c3 = b18
symbol c4 = b19
symbol c5 = b20
symbol c6 = b21
symbol c7 = b22
symbol c8 = b23
goto main

update:
'update the motors
HI2COUT 1,(150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150,
return

main:

goto main
```

This initiates the program and the I2C bus and then defines the symbols for easier reference. Default variables in PICAxe are numbered from b0-b27. This meant that I used d for axis b, to avoid a conflict.

The update subroutine simply updates the SD20 with the variables. This code centres the servos, for ease of programming- The final version would have variable labels instead of "...150, 150, 150…"

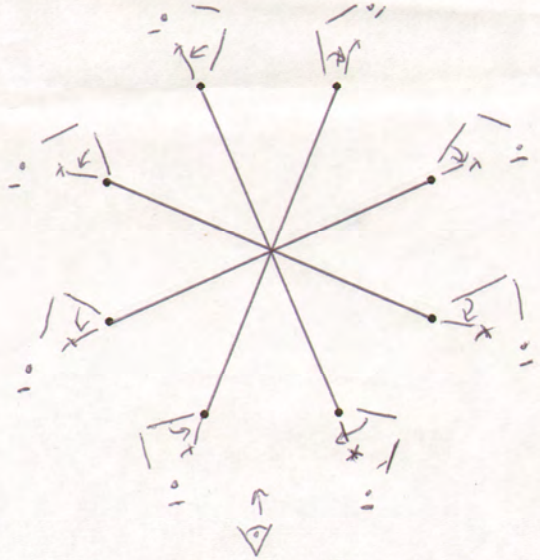

A:
Each page can be split into 6 sections. The top row is the first phase and the bottom row is the second
1:
B: How to read the diagrams
Each column is for one axis. Often, an axis in each phase is blank
C:
Position of "foot"
Position of "knee"
"knee" and "foot" are inline
2:
Shows a weightbearing, inline leg
Axis A: Centered. Arrows denote movement
Axis B: Shows elevation of upper and/or lower leg. NOT Position
The natural standing position:


Axis A
Stand up from reset pattern
Axis B
Axis C
Legs centre and simultaneously rise into the air.
This reduces the chance of the lower leg snagging on another or an external object.
Phase 1:
The lower leg slides downwards until it is perpendicular to the ground.
At this point, it becomes weight bearing.
Phase 2:

A

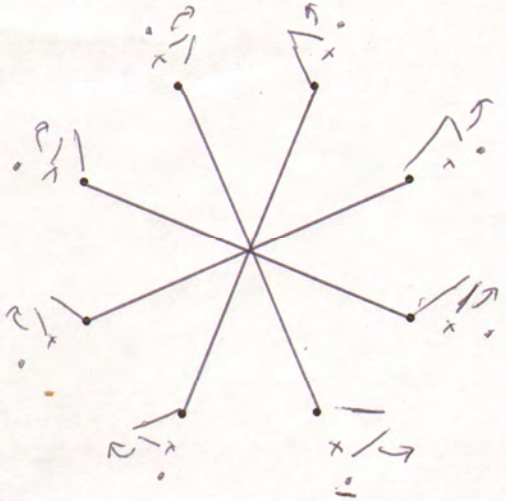
The upper leg moves towards a central position, likewise, the lower leg does. It doubles back on itself to remain weight bearing.

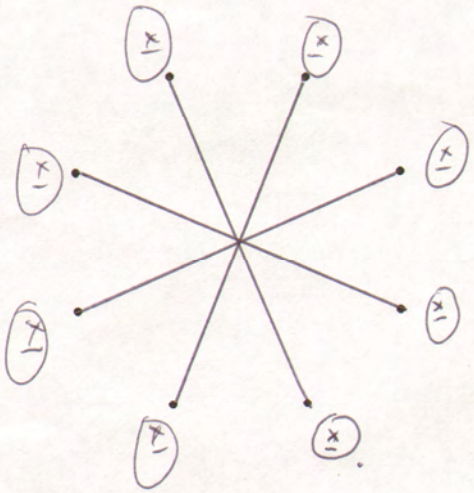
B



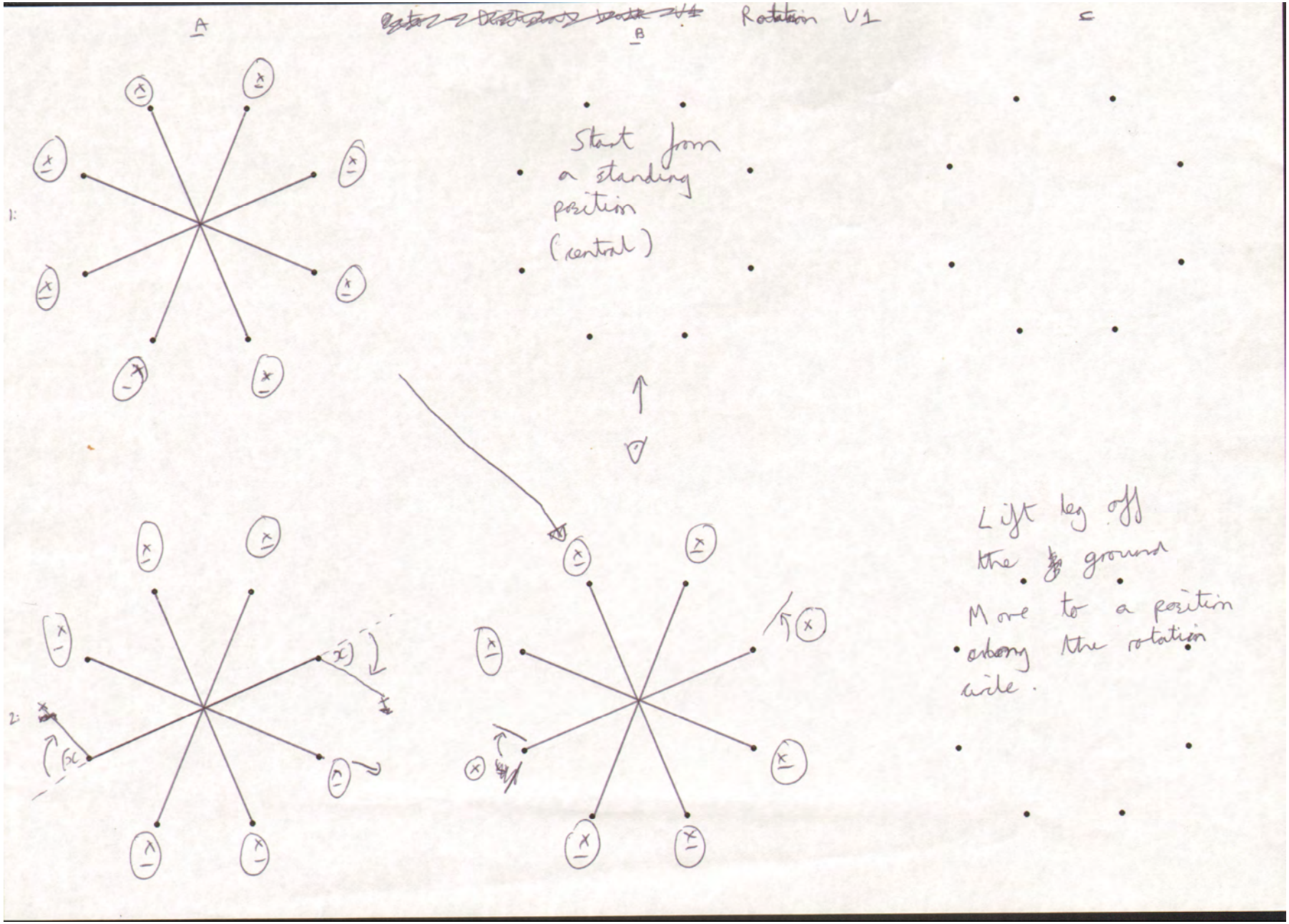
Terminating Position

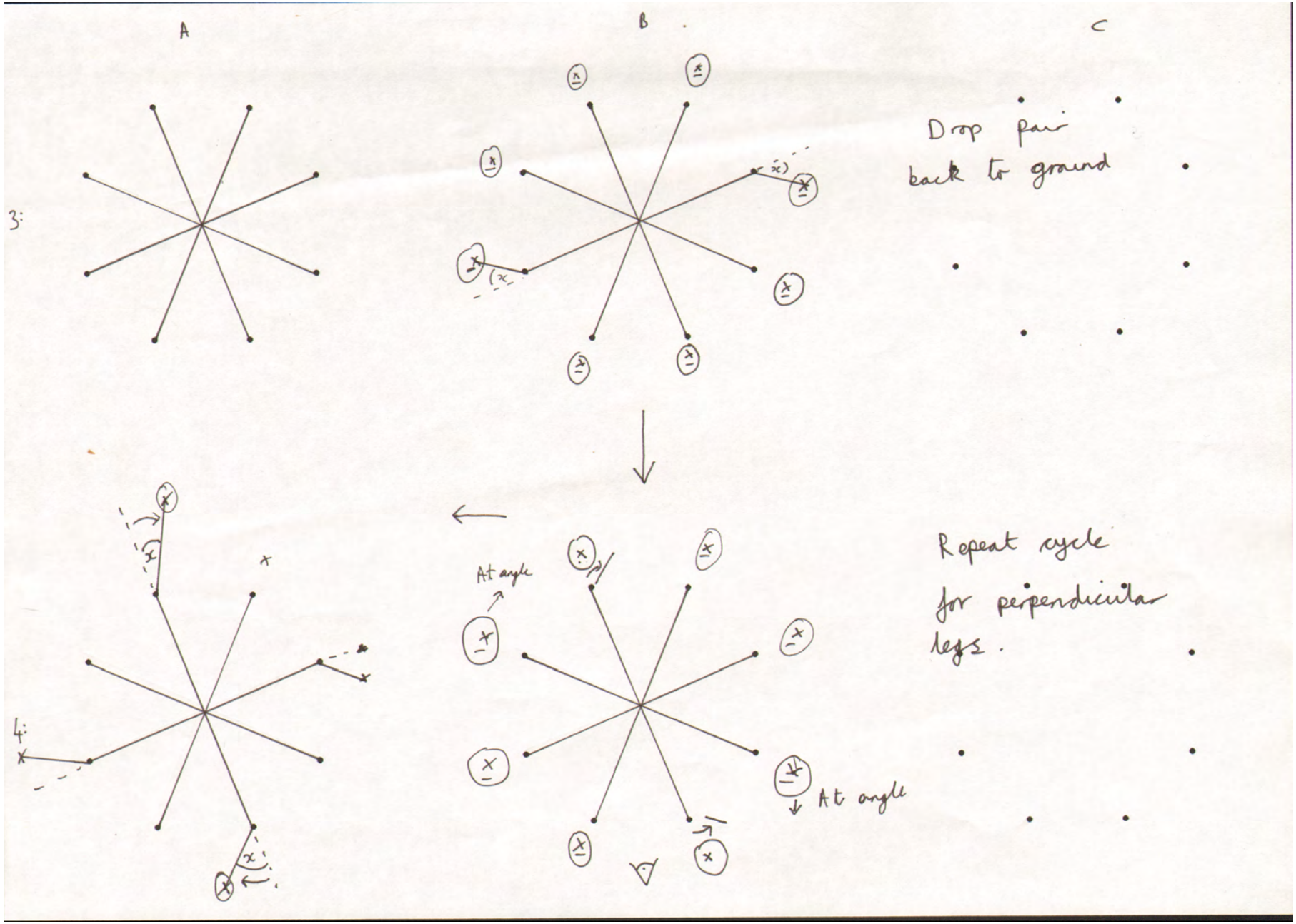
C

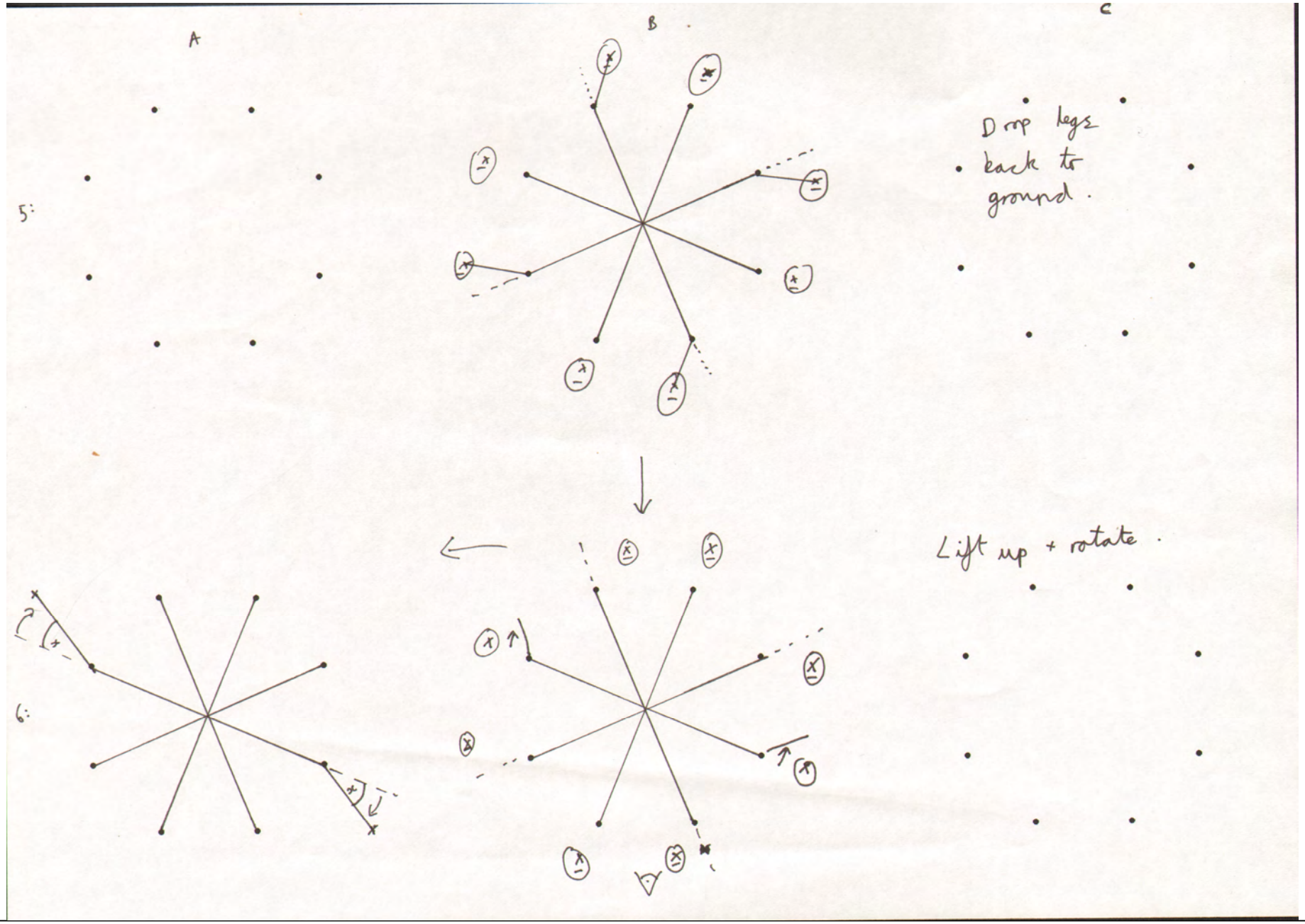


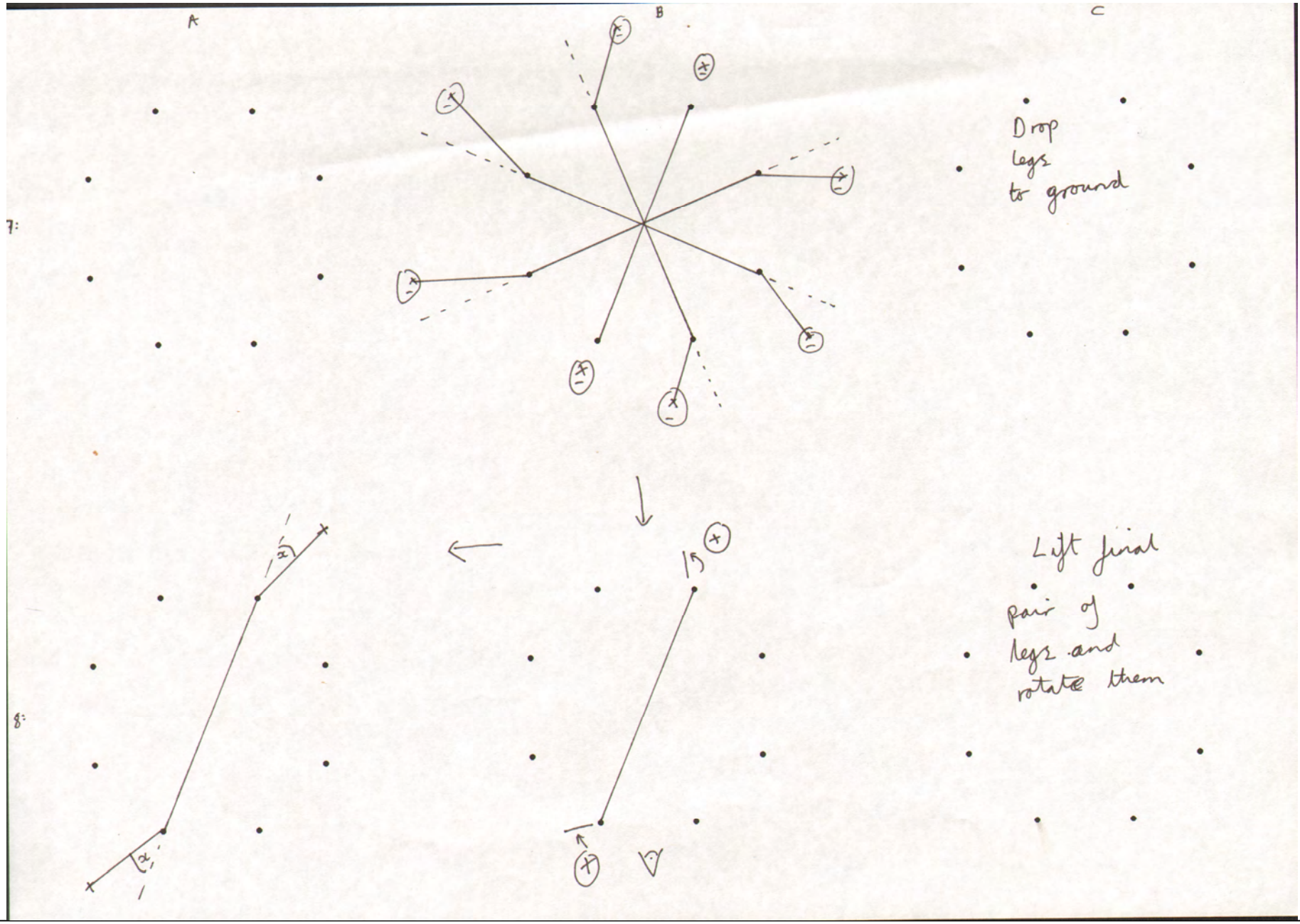


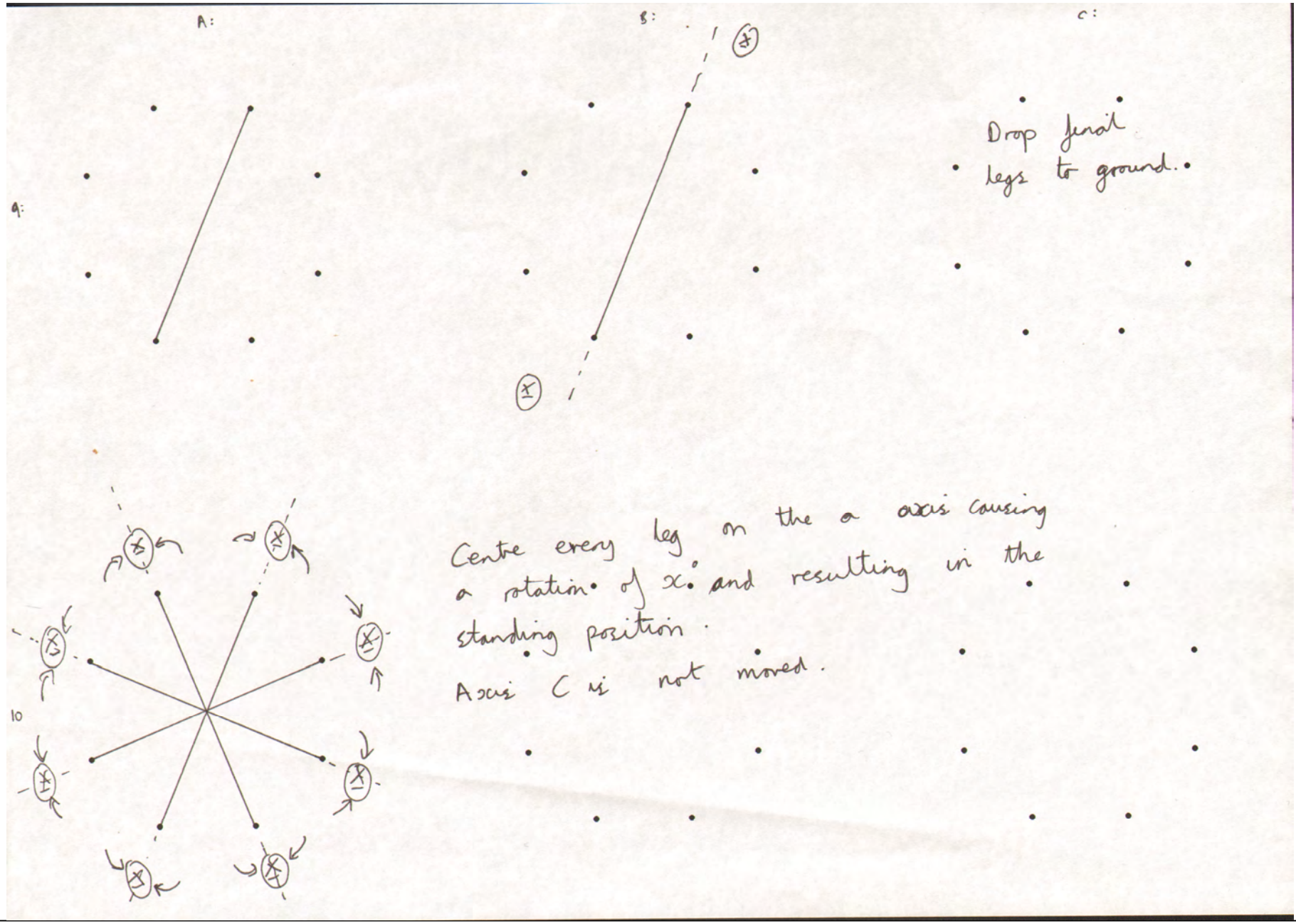
Every servo is central

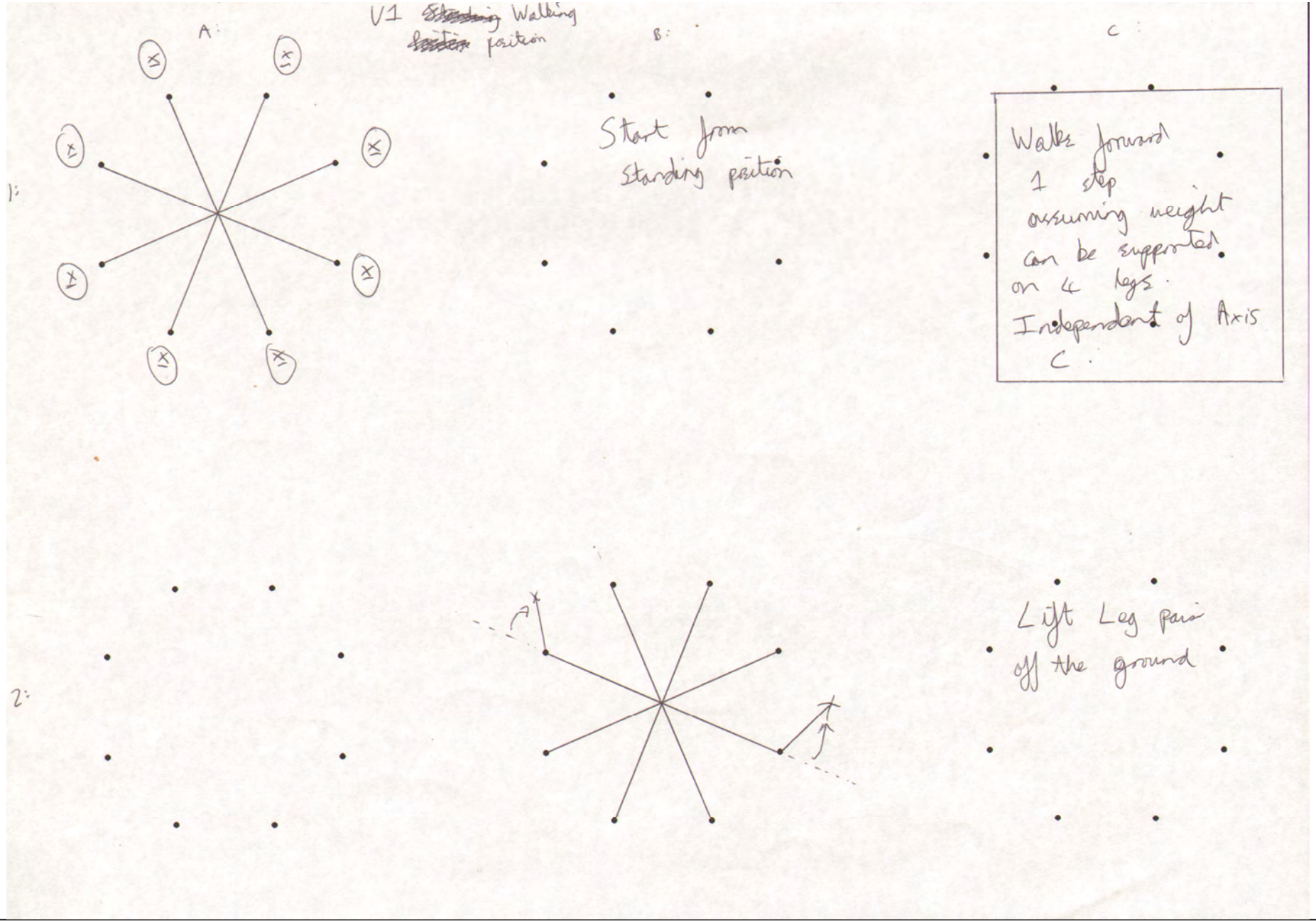

A
B
Rotation V1
Start from a standing position (central)
Lift leg off the ground
Move to a position along the rotation axle.


A
B
C
3:
Drop pair back to ground
4:
At angle
Repeat cycle for perpendicular legs.
At angle

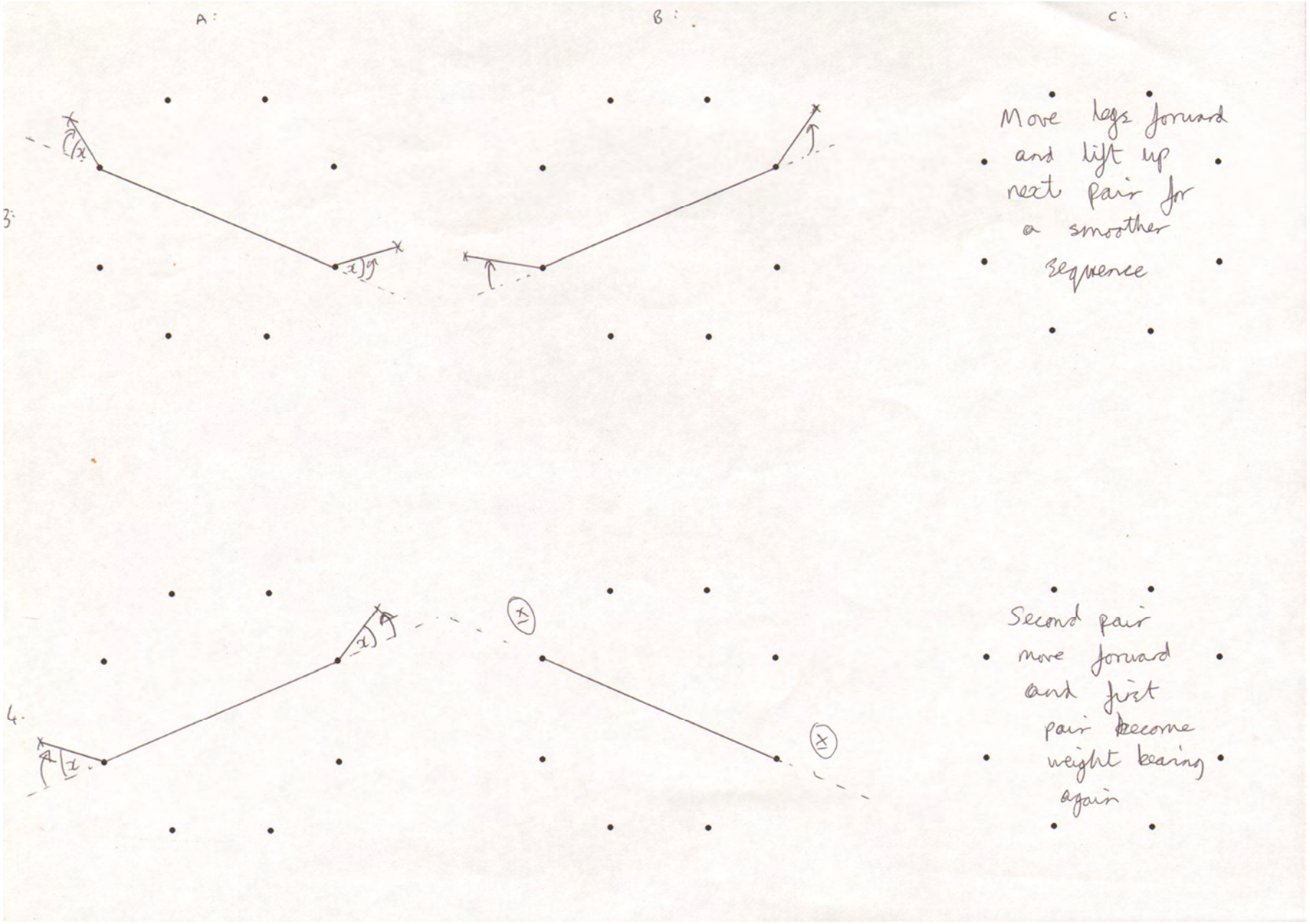

A
B
C
5:
Drop legs back to ground.
Lift up + rotate.
6:

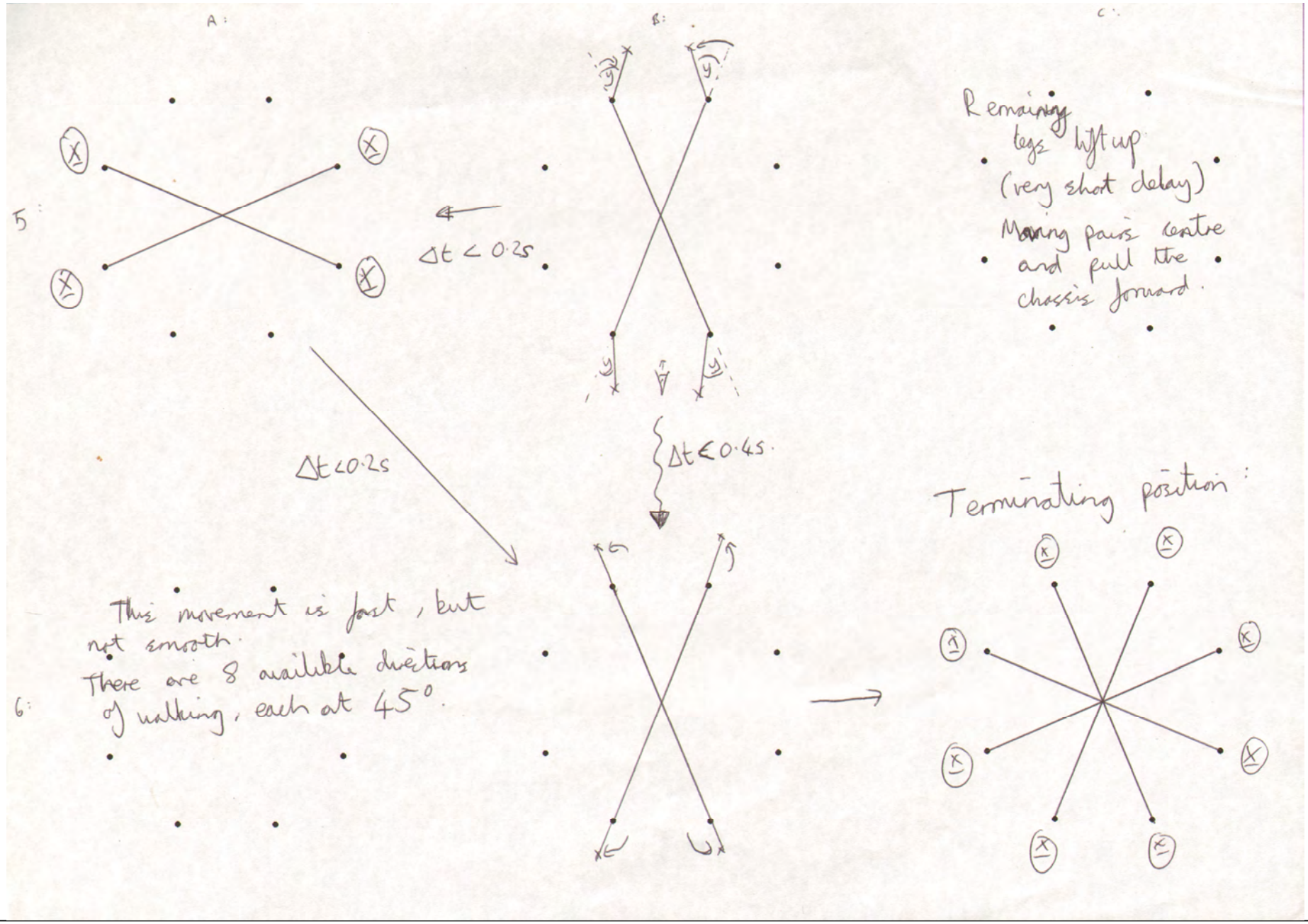

A
B
C
7:
Drop
legs
to ground
8:
Lift finial
pair of
legs and
rotate them

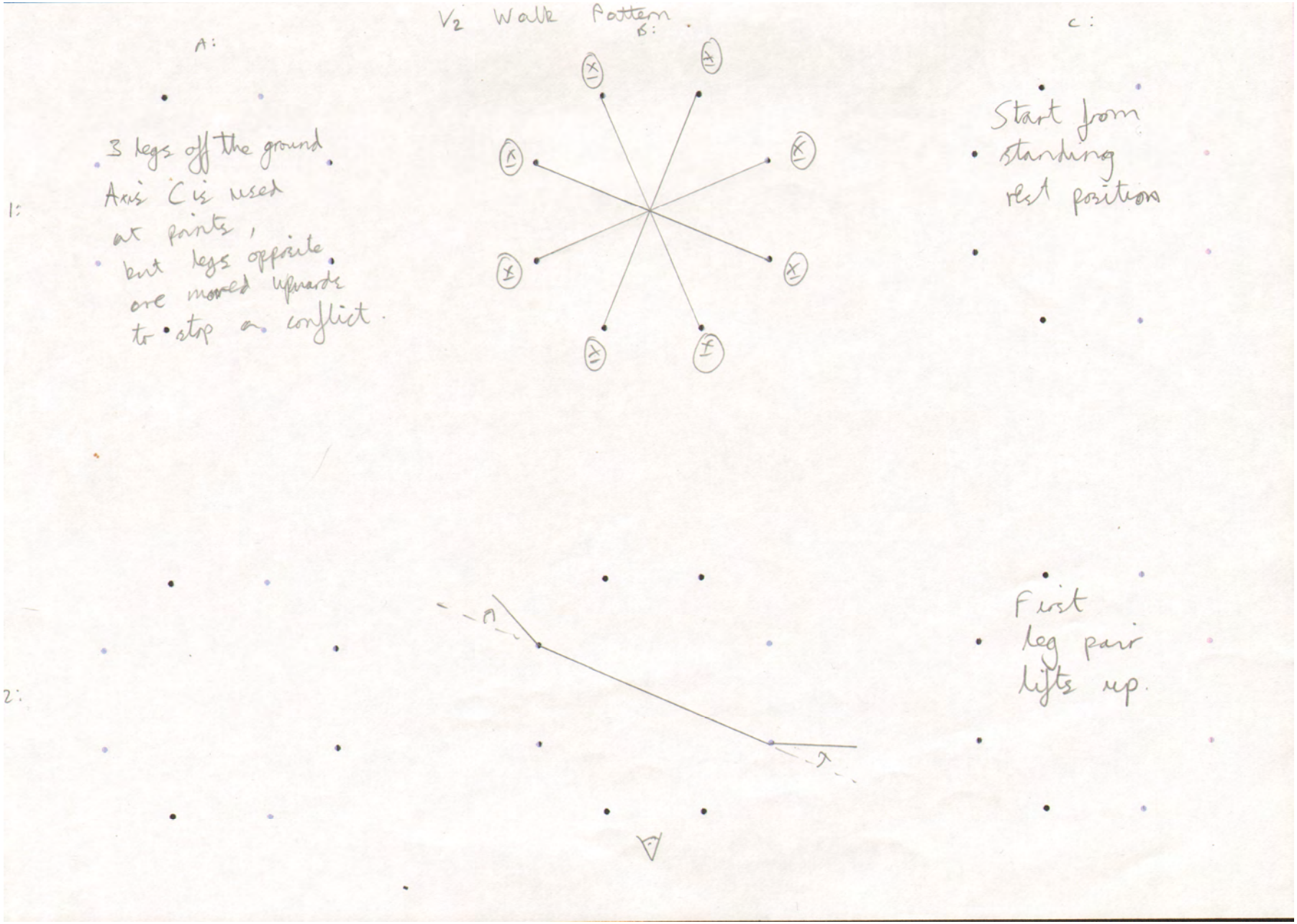

A:
B:
C:
Drop final legs to ground.
9:
Centre every leg on the a axis causing a rotation of x° and resulting in the standing position.
Axis C is not moved.
10


V1 ~~Standing~~ Walking ~~Sitting~~ position
A:
B:
C:
1:
2:
Start from standing position
Walks forward 1 step assuming weight can be supported on 4 legs. Independant of Axis C.
Lift Leg pair off the ground

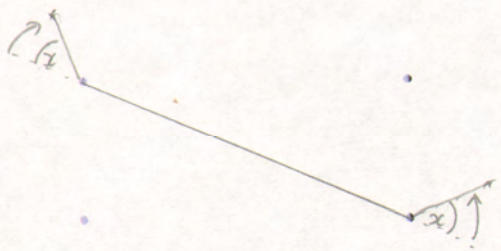
Section 4 – Making - Programming -Leg Diagrams




A:
B:
C:
5:
Δt < 0.2s
Δt < 0.2s
Δt ∈ 0.4s.
Remaining legs lift up (very short delay)
Moving pairs centre and pull the chassis forward.
Terminating position:
This movement is fast, but not smooth.
There are 8 available directions of walking, each at 45°.
6:


V2 Walk Pattern
A:
B:
C:
3 legs off the ground
Axis C is used at points, but legs opposite are moved upwards to stop a conflict.
Start from standing rest position
1:
2:
First leg pair lifts up.

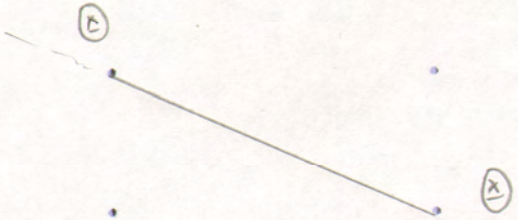
A:



NB: It is vital x remains constant in this example

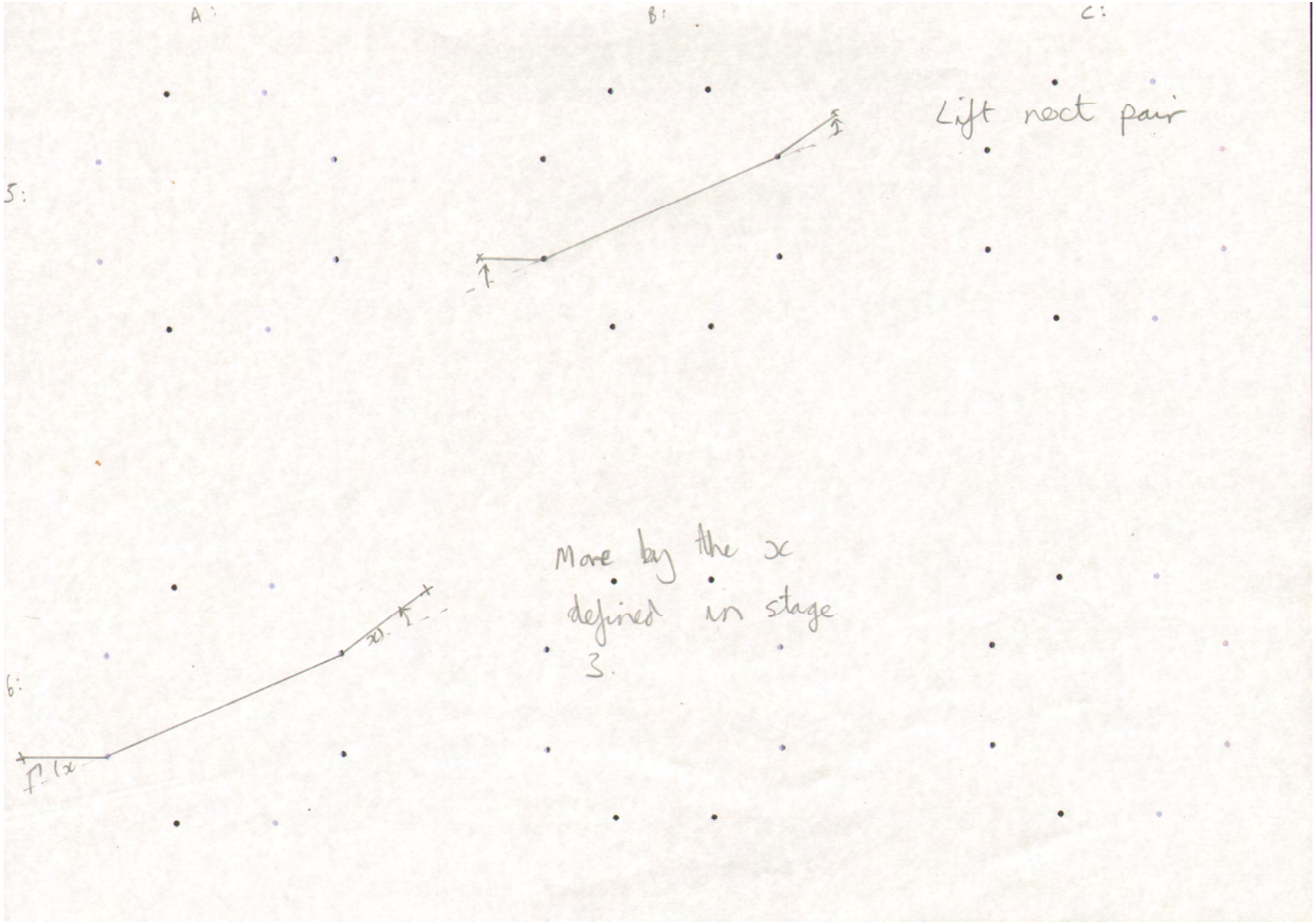
B:

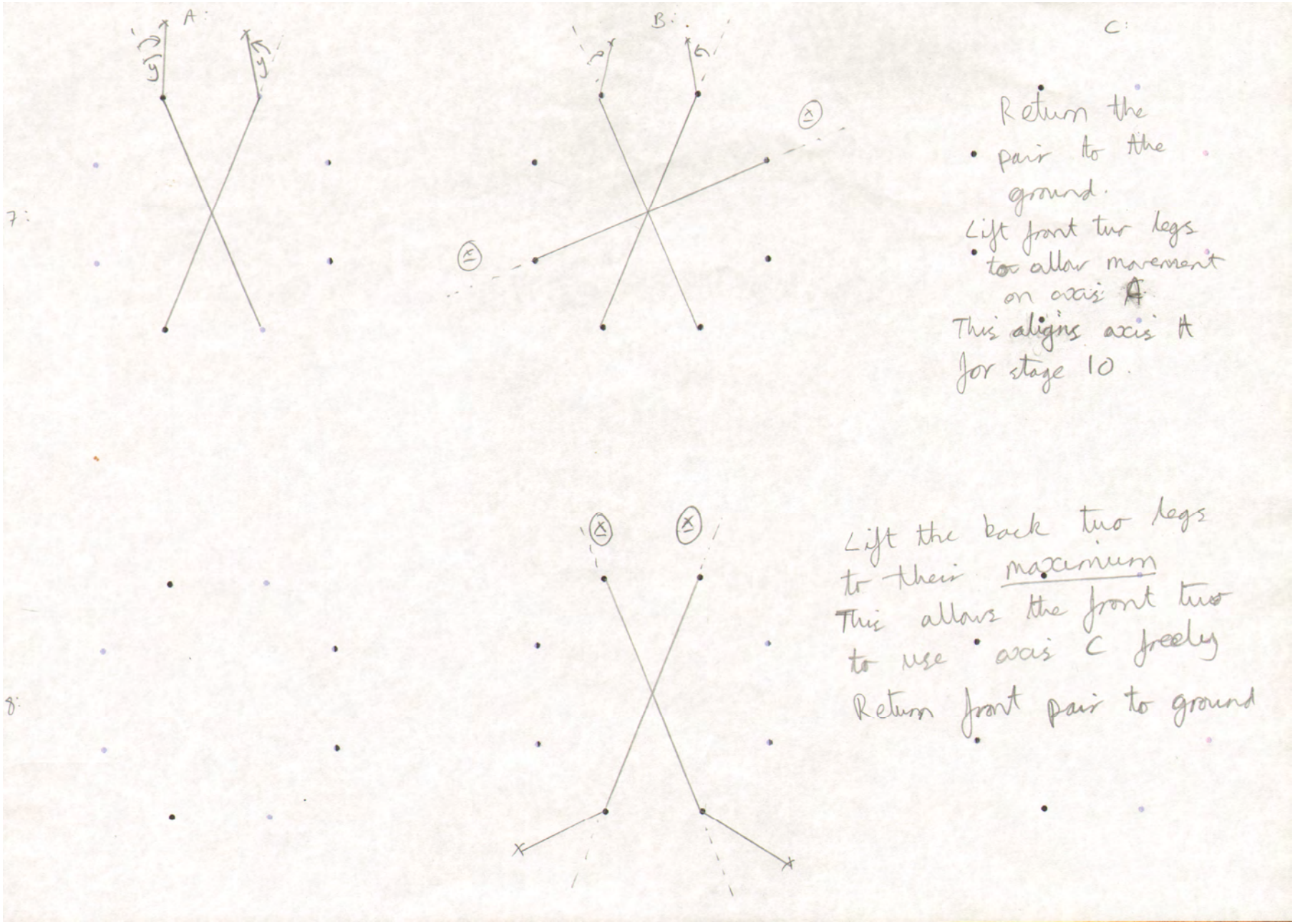
More legs forward



C:

NB: Steps 2 and 4 can be performed simultaneously, followed by steps 3 and 5. This means that 4 legs are off the ground, however


A:
B:
C:
Lift next pair
5:
Move by the x
defined in stage
3.
6:


A:
B:
C:
7:
8:
Return the pair to the ground.
Lift front two legs to allow movement on axis A
This aligns axis A for stage 10.
Lift the back two legs to their maximum
This allows the front two to use axis C freely
Return front pair to ground

A:

9:

- Lower and stretch out the front two legs
- Can be split into stages to keep legs weight bearing

B:

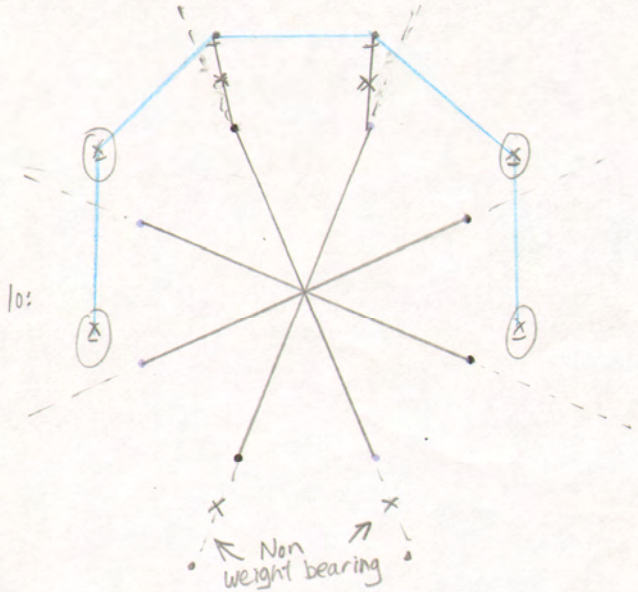
Momentarily before axis B.

C:

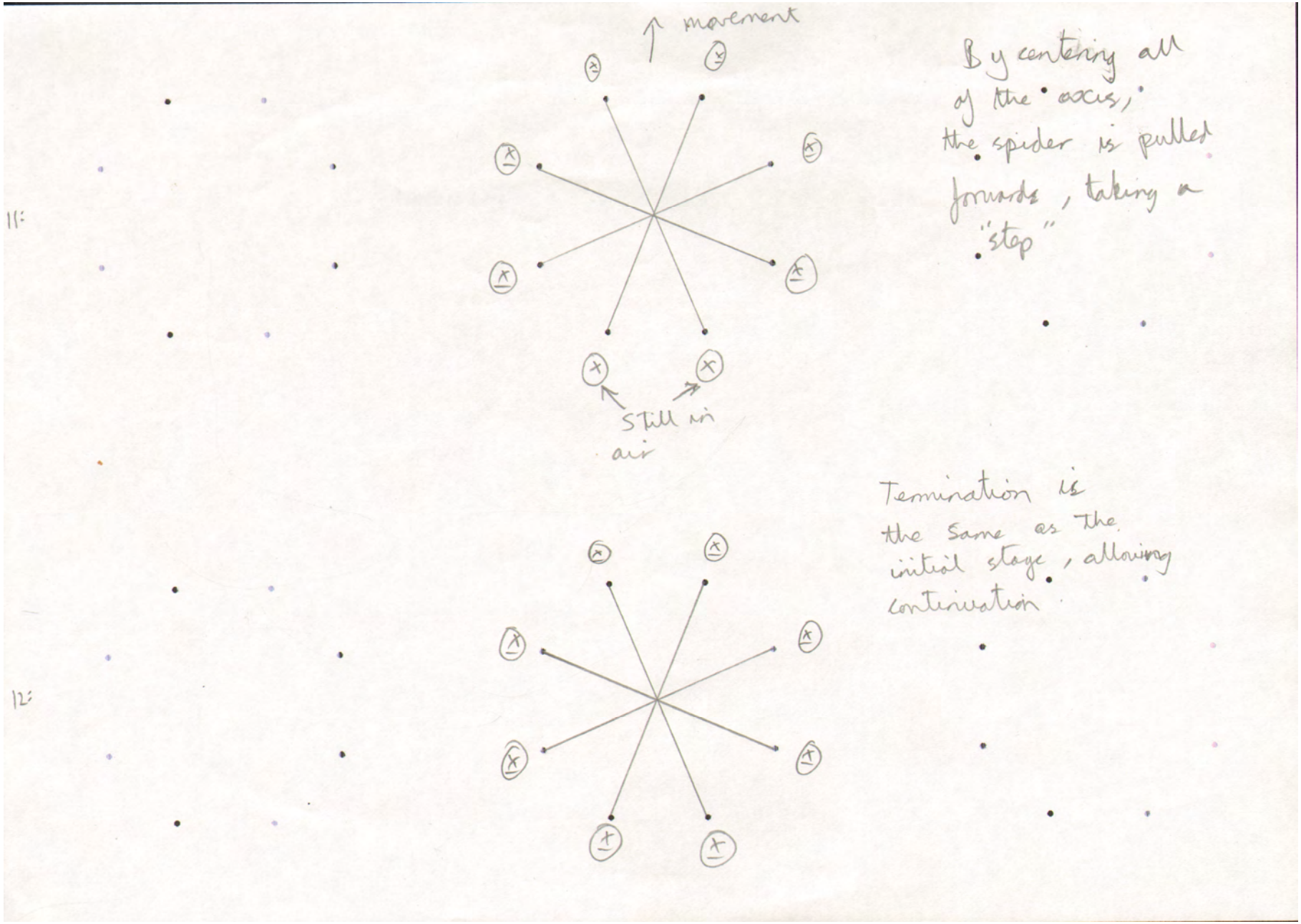
(downwards)

- This results in this position:

10:



The weight bearing legs show the start of a perfect octagon.


movement
By centering all of the axis, the spider is pulled forwards, taking a "step"
Still in air
Termination is the same as the initial stage, allowing continuation

Standing program:
It is important to note that the servos only actually move when update is called.

```
'
'Program for main robot board
'
init:
'Initialise chip
#no_table
#no_data
'Start i2c
HI2CSETUP I2CMASTER, $C2, i2cfast, i2cbyte
'Define symbols for ease of use
symbol a1 = b0
symbol a2 = b1
symbol a3 = b2
symbol a4 = b3
symbol a5 = b4
symbol a6 = b5
symbol a7 = b6
symbol a8 = b7
symbol d1 = b8
symbol d2 = b9
symbol d3 = b10
symbol d4 = b11
symbol d5 = b12
symbol d6 = b13
symbol d7 = b14
symbol d8 = b15
symbol c1 = b16
symbol c2 = b17
symbol c3 = b18
symbol c4 = b19
symbol c5 = b16
symbol c6 = b17
symbol c7 = b18
symbol c8 = b19
'Sensor symbols
symbol x = b24
symbol y = b25
symbol z = b26
symbol gps = b27
'This needs a word as it can be greater than 255
symbol standard = w11
'Set central positions
symbol standard = w11
'Set central positions
a1 = 127
a2 = 127
a3 = 127
a4 = 127
a5 = 127
a6 = 127
a7 = 127
a8 = 127
d1 = 127
d2 = 127
d3 = 127
d4 = 127
d5 = 127
d6 = 127
d7 = 127
d8 = 127
c1 = 127
c2 = 127
c3 = 127
c4 = 127
c5 = 127
c6 = 127
c7 = 127
c8 = 127
'b20 and b21 are unused so far

'Set standard pause time between movements
standard = 1000
'Update the legs to centre them
gosub update
'Pause for standard time
pause standard
goto main

update:
'Raw data for offsets:
'HI2COUT 1,(129,127,140,121,127,
'137,127,127,127,122,
'116,141,127,127,100,
'127,122,127,132,127)
```

```
'This adds in the offsets
a1 = a1-5
a2 = a1+13
a3 = a3
a4 = a4
a5 = a5
a6 = a6-11
a7 = a7
a8 = a8
d1 = d1+2
d2 = d2-6
d3 = d3+14
d4 = d4
d5 = d5+10
d6 = d6
d7 = d7-27
d8 = d8+5
c1 = c1
c2 = c2-5
c3 = c3
c4 = c4
c5 = c5
c6 = c6-5
c7 = c7
c8 = c8
'Update the motors
HI2COUT 1,(d1,c1,a2,d2,a3,d5,a4,a5,d4,c2,a6,d3,d6,a7,d7,c3,a1,c4,d8,a8)
'Remove offsets
a1 = a1+5
a2 = a1-13
a3 = a3
a4 = a4
a5 = a5
a6 = a6+11
a7 = a7
a8 = a8
d1 = d1-2
d2 = d2+6
d3 = d3-14
d4 = d4
d5 = d5-10
d6 = d6
d7 = d7+27
d8 = d8-5
c1 = c1
```

```
128   c2 = c2+5
129   c3 = c3
130   c4 = c4
131   c5 = c5
132   c6 = c6+5
133   c7 = c7
134   c8 = c8
135   'Update accelerometer data
136   gosub accel
137   return
138
139   accel:
140   'Read all the axis
141   readadc 0, z
142   readadc 1, y
143   readadc 2, x
144   'Output the data
145   serout 7, T4800_8, ("UUU",x,",",y,",",z)
146   return
147
148   main:
149   'Wait for a command from the controller
150   serin 1,T4800_4, b20
151   'Determine the ASCII values of the data (note: not binary)
152   if b20 = 49 then goto forward1
153   if b20 = 50 then goto acw
154   if b20 = 51 then goto cw
155   if b20 = 52 then goto back1
156   goto main
157
158   'Intermediate stages follow
159   forward1:
160   gosub forward_walk
161   goto main
162
163   acw:
164   gosub rotate_acw
165   goto main
166
167   cw:
168   gosub rotate_cw
```

```
169   goto main
170
171   back1:
172   gosub backward_walk
173   goto main
174
175   'A routine to move all of axis A
176   testa:
177   a1 = 30
178   a2 = 30
179   a3 = 30
180   a4 = 30
181   a5 = 30
182   a6 = 30
183   a7 = 30
184   a8 = 30
185   gosub update
186   pause standard
187   a1 = 225
188   a2 = 225
189   a3 = 225
190   a4 = 225
191   a5 = 225
192   a6 = 225
193   a7 = 225
194   a8 = 225
195   gosub update
196   pause standard
197   return
198
199   'A routine to move all of axis B
200   testb:
201   d1 = 30
202   d2 = 30
203   d3 = 30
204   d4 = 30
205   d5 = 30
206   d6 = 30
207   d7 = 30
208   d8 = 30
```

```
209   gosub update
210   pause standard
211   return
212
213   'A routine to move all of axis C
214   testc:
215   c1 = 30
216   c2 = 30
217   c3 = 30
218   c4 = 30
219   c5 = 30
220   c6 = 30
221   c7 = 30
222   c8 = 30
223   gosub update
224   pause standard
225   c1 = 225
226   c2 = 225
227   c3 = 225
228   c4 = 225
229   c5 = 225
230   c6 = 225
231   c7 = 225
232   c8 = 225
233   gosub update
234   pause standard
235   return
236
237   'A routine to centre the legs
238   centre:
239   a1 = 127
240   a2 = 127
241   a3 = 127
242   a4 = 127
243   a5 = 127
244   a6 = 127
245   a7 = 127
246   a8 = 127
247   d1 = 127
248   d2 = 127
249   d3 = 127
250   d4 = 127
```

```
251   d5 = 127
252   d6 = 127
253   d7 = 127
254   d8 = 127
255   c1 = 127
256   c2 = 127
257   c3 = 127
258   c4 = 127
259   c5 = 127
260   c6 = 127
261   c7 = 127
262   c8 = 127
263   gosub update
264   pause standard
265   return
266
267   rotate_cw: 'Follows the "rotate v1" pattern
268   d1 = 90
269   d5 = 90
270   gosub update
271   pause standard
272   a1 = 100
273   a5 = 100
274   gosub update
275   pause standard
276   d1 = 127
277   d5 = 127
278   gosub update
279   pause standard
280   d3 = 90
281   d7 = 90
282   gosub update
283   pause standard
284   a3 = 100
285   a7 = 100
286   gosub update
287   pause standard
288   d3 = 127
289   d7 = 127
290   gosub update
291   pause standard
```

```
292   d2 = 90
293   d6 = 90
294   gosub update
295   pause standard
296   a2 = 100
297   a6 = 100
298   gosub update
299   pause standard
300   d2 = 127
301   d6 = 127
302   gosub update
303   pause standard
304   d4 = 90
305   d8 = 90
306   gosub update
307   pause standard
308   a4 = 100
309   a8 = 100
310   gosub update
311   pause standard
312   d4 = 127
313   d8 = 127
314   gosub update
315   pause standard
316   gosub centre
317   return
318
319   rotate_acw: 'Follows the "rotate v1" pattern
320   d1 = 90
321   d5 = 90
322   gosub update
323   pause standard
324   a1 = 164
325   a5 = 164
326   gosub update
327   pause standard
328   d1 = 127
329   d5 = 127
330   gosub update
331   pause standard
332   d3 = 90
```

```
333   d7 = 90
334   gosub update
335   pause standard
336   a3 = 164
337   a7 = 164
338   gosub update
339   pause standard
340   d3 = 127
341   d7 = 127
342   gosub update
343   pause standard
344   d2 = 90
345   d6 = 90
346   gosub update
347   pause standard
348   a2 = 164
349   a6 = 164
350   gosub update
351   pause standard
352   d2 = 127
353   d6 = 127
354   gosub update
355   pause standard
356   d4 = 90
357   d8 = 90
358   gosub update
359   pause standard
360   a4 = 164
361   a8 = 164
362   gosub update
363   pause standard
364   d4 = 127
365   d8 = 127
366   gosub update
367   pause standard
368   gosub centre
369   return
370
371
372   forward_walk: 'Follows the "walk v2" pattern
373   d1 = 90
374   d5 = 90
```

```
375   gosub update
376   pause standard
377   a1 = 90
378   a6 = 164
379   gosub update
380   pause standard
381   d1 = 127
382   d5 = 127
383   gosub update
384   pause standard
385   d8 = 90
386   d4 = 90
387   gosub update
388   pause standard
389   a8 = 90
390   a4 = 164
391   gosub update
392   pause standard
393   d8 = 127
394   d4 = 127
395   gosub update
396   pause standard
397   d6 = 90
398   d2 = 90
399   d7 = 90
400   d3 = 90
401   gosub update
402   pause standard
403   c2 = 20
404   c3 = 20
405   d2 = 164
406   d3 = 164
407   gosub update
408   pause standard
409   gosub centre
410   return
411
412   stablise: 'Stop the robot slipping downwards after movements
413   d1 = 90
414   d5 = 90
415   pause standard
```

```
416   gosub centre
417   c1 = 160
418   pause standard
419   gosub centre
420   d1 = 127
421   d5 = 127
422   c1 = 127
423   pause standard
424   gosub centre
425   d2 = 90
426   d6 = 90
427   pause standard
428   gosub centre
429   c2 = 160
430   pause standard
431   gosub centre
432   d2 = 127
433   d6 = 127
434   c2 = 127
435   pause standard
436   gosub centre
437   d3 = 90
438   d7 = 90
439   pause standard
440   gosub centre
441   c3 = 160
442   pause standard
443   gosub centre
444   d3 = 127
445   d7 = 127
446   c3 = 127
447   pause standard
448   d4 = 90
449   d8 = 90
450   pause standard
451   gosub centre
452   c4 = 160
453   pause standard
454   gosub centre
455   d4 = 127
456   d8 = 127
```

```
457   c4 = 127
458   pause standard
459   return
460
461   backward_walk: 'Follows the "walk v2" pattern
462   d1 = 90
463   d5 = 90
464   gosub update
465   pause standard
466   a1 = 164
467   a6 = 90
468   gosub update
469   pause standard
470   d1 = 127
471   d5 = 127
472   gosub update
473   pause standard
474   d8 = 90
475   d4 = 90
476   gosub update
477   pause standard
478   a8 = 164
479   a4 = 90
480   gosub update
481   pause standard
482   d8 = 127
483   d4 = 127
484   gosub update
485   pause standard
486   d6 = 90
487   d2 = 90
488   d7 = 90
489   d3 = 90
490   gosub update
491   pause standard
492   c2 = 20
493   c3 = 20
494   d2 = 164
495   d3 = 164
496   gosub update
497   pause standard
498   gosub centre
499   return
```

Section 5 – Testing and Evaluation - Instructions for use

When the controller is wired to the robot:

Turn the robot on at the mains, through the computer power supply and then put a 4.5V battery on the battery clip of the main board. Then turn the controller on by the switch on the base of one of the faces.

The joystick controls all movement on one axis. To walk forwards, move it all the way forwards and to walk backwards, move it all the way backwards. To rotate clockwise, move it halfway backwards, and to rotate clockwise, move it halfway forwards.

To operate the camera, plug a 9V battery on the camera battery clip, and the receiver into the mains and a screen. Tune the camera by moving the tuning knob until there is a picture.

The robot will not make a new movement until it has completed the last one. It is important that the legs are positioned safely, to avoid damaging the servos.

Problem Solving

My entire project was based around a constant stream of problems that needed solving. The weight, power, strength, equipment, system control (etc.) were all problems which needed solving in a unique way to create a unique end product.

Section 5 – Testing and Evaluation - Cost Analysis

Prototyping costs:				
Item:	Cost:	Quantity:	Shipping:	Total:
Servos	5.99	27	7.9	169.63
	5.99	**27**	**7.9**	**169.63**
Final design costs:	**Total:**			
Item:	Cost:	Quantity:	Shipping:	Total:
Servos	13.25	24	0	318
Servo Driver	8.08	2	2.5	18.66
Servo Extension leads	1.45	4	3.2	9
Board	74	1	8.14	82.14
GPS + Antenna	44.13	1	5.97	50.1
Accelerometer	15.98	1	0	15.98
Radios + Antennas	68.45	1	0	68.45
Perspex Cube (controller)	6.43	1	0	6.43
Camera	16.48	1	0	16.48
Perspex Tube (outer case)	20	1	0	20
High Current Wire	1.19	8	0	9.52
	225.34	**34**	**19.81**	**614.76**
	Total:	**784.39**		
I have not included costs under £5 for screws, fixings and similar items.				
I have also not included the cost of school provided materials, such				
as MDF and copper clad board.				
Costs of tools has also been omitted due to reusability, and I have				
ignored items I sourced for free.				

The cost of the project has been the major downside, other that the problems I had with weight.

This is a shame because it is a unique end product, and one with could prove to be very useful in the right hands.

Bulk buying these items would greatly reduce the cost- if it were to be mass produced.

Section 5 – Testing and Evaluation - Comparison of Product and Specification

Initial Specification

- The robot should be remotely operated.
- It should be capable of sending back information, including a camera feed, data about the surroundings and navigational aids.
- It should be able to cover all terrain, including rough ground such as forests and severe inclines.
- Steps may need to be overcome, so there should be a mechanism to allow this.
- It should have enough power such that it is never left stranded.
- It should be light so that it can't cause any damage to an already devastated area and it can move subtly.
- It must be fast and agile; able to manoeuvre in enclosed spaces.
- There should be a way of manufacturing it in an industrial environment.
- It must have reliable operation.
- The controls must be simple to use but also versatile and powerful.
- The wireless link between device and controller must be reliable.
- It should be made from parts which are easily sourced, in case of damage.
- Water resistivity would be incredibly useful.
- If possible, a microphone output should be also send back.
- If it runs out of power, it must have a backup system to stop it damaging itself or surroundings.
- Reliable and powerful navigational aids should be on board, otherwise it is no use for surveying.
- To assist with the surveying, it must carry equipment to monitor the environment.
- It must be fairly hardy and able to withstand minor damage
- The ground it moves along might well be wet or slippery, and it must overcome this.
- It must not be possible to disorient the moving mechanisms, ie, falling over, rolling etc.

There are no points which I set out to achieve and completely failed, and only three which I did not fully achieve- two of these could be added on with just programming extra routines. I think this shows that I achieved what I set out to achieve.

On reflection, planning my time before I really had any understanding of the project I was undertaking was pointless. The time plan was a brief idea of how I wanted to spend the time and set some deadlines. I quickly discovered that these stages had to be completed in unison, as everything depended on each other. My deadlines were, on the whole, kept to, however I feel that the time keeping could have been better managed.

Evaluation

I have succeeded in creating a unique, useful and fun robot, with serious potential for actual use. Compared to small vehicles of a similar specification, it is very cheap, and it is truly all terrain. The materials are easy to source, and it has the potential to be made on an industrial scale.

Making the robot was truly enjoyable to create. I learnt many new skills, about new technology, and a lot about actual making of things, rather than just electronics. The project includes advanced electronics, advanced programming techniques and some complex concepts that I would never have dreamt of using otherwise.

Much of the prototyping I did I didn't have a chance to fully utilise in the end product- things like the computer control. This leaves a lot of options for future development. I would be very interested to expand the robot in the future, it would certainly be feasible.

I hope that legged vehicles continue to be developed by students, and in industry, as they have great potential for military applications, space exploration and the surveying of disaster zones- I hope I have proved this in my work.

The robot and I gained a place in the National Science Competition 2011 Finals, where it was Highly Commended in the Senior Engineering category (equivalent to 3rd in the country).

I also won the Intel prize, which was a place to represent the UK in the International Science Competition Finals in Los Angeles in May.

At the International Science and Engineering Fair, I won a $25,000 Scholarship from Agilent, one of the highest awards. I also ranked somewhere in the top 31 for a systems engineering award, and was highly commended by the US Army, Navy, IEEE, IEEE Computer Science Department and many other.

Appendix 1 - Email Correspondence

Sparkfun (Accelerometer board)

Me -> Sparkfun

Hi,

I'm a student studying at Sutton Grammar School for Boys in the UK.

For my current electronics project, I am creating a Robotic Spider for use as a remote surveillance vehicle. Details of my project can be found here and I have attached the same document. Photos can be found here (*hyperlink to some photos)*.

To enhance the functionality, I am looking to add some accelerometers. Your product SEN-00849 fulfils my criteria perfectly and I was wondering if you might consider sponsoring me a few units of this item in return for me featuring your logo on both my project folder and on my presentations at approaching science and engineering competitions.

Thanks,

Tom.

Sparkfun -> Me

Hi Tom,

Thank you for sharing your project! It looks really amazing, and I've forwarded it on to our Marketing Department. Because our company is based around offering the lowest price possible for all our items, we do not offer any discounts or samples, but we'll let Marketing make the final decision on this. They may ask you if we can feature your project on a home page post, but again, I'll leave that up to them.

Anyway, thanks for sharing, and good luck with your project!
Regards,

Fran O'Rourke
Customer Service
SparkFun Electronics
303-284-0979

Me -> Sparkfun

Thank you very much, I'll email you some photos of the final version when I get there.

Thanks,

Tom.

Sparkfun -> Me

Great Tom! We'll look forward to seeing them.

Regards,

Fran O'Rourke
Customer Service
SparkFun Electronics
303-284-0979

Appendix 1 - Email Correspondence

Novarm (Diptrace)

Me -> Novarm

Hi,
I've been working on a robotic spider project since the beginning of September at Sutton Grammar School for Boys, and it is just starting to come together.
I have some photos here if you would like to have a look: http://picasaweb.google.com/tladyman

However, I am currently in the process of remaking the electronics, but I have too many pins to use the Freeware license. I wonder if you could please send me a Non-Profit Product Key?

Thanks,
Tom Ladyman

Novarm -> Me

Dear Tom,

Use the following registration data

User name: Non-profit use only

Reg. key: NM59-S89M-99X6-NNGM

Registration data should be copied into Help/Register dialog box in Freeware
or into nag-screen in trial version.

Notice that Non-Profit (Lite) license works only starting from 2.0.06, so if you have earlier version please download update from http://www.diptrace.com/download.php

With kind regards,
Victor Savenko
DipTrace Team

Me -> Novarm

Thank you very much!