Integrated Design Project

First Report

The El Thetas

Robot Name Here

Group L108

Adam Greig	Selwyn College	Lab Group 80
Jon Sowman	Selwyn College	Lab Group 80
Stephen Kerr	Corpus Christi College	Lab Group 101
Peter Taylor	Corpus Christi College	Lab Group 101
James Goodwin	Robinson College	Lab Group 85
Vasilis Ioannou	Robinson College	Lab Group 85

Solution Neutral Problem Statement

Collect and sort coloured bobbins from a rack, placing one bobbin of each colour into a container, then deliver filled containers to a delivery zone.

Requirements

1. Solution should be an autonomous guided vehicle (AGV)
2. The AGV must stay within the playing area
3. The AGV must be capable of following lines to reach objectives
4. The AGV must be able to detecting, picking up and carrying bobbins
5. The AGV must be capable of depositing the bobbins in boxes
6. The AGV must deposit two bobbins in each box
7. The AGV must ensure each box has one bobbin of each colour before depositing
8. The AGV must be able to detect, pick up, carry and deposit boxes
9. The boxes must be able to contain three bobbins
10. The AGV must be able to identify the initial bobbin colour in the box
11. The AGV must be able to identify the colour of the currently picked up bobbin
12. The AGC must indicate the identified bobbin colour using three LEDs
13. The AGV must be able to navigate between the boxes, the bobbins and the box delivery zone
14. The AGV should be able to detect and recover from collisions
15. The AGV should be capable of recovering from losing a line
16. The AGV should be capable of detecting a bad bobbin identified by a metal insert and top disk
17. The AGV should be capable of pushing the bad bobbin off the bobbin rack

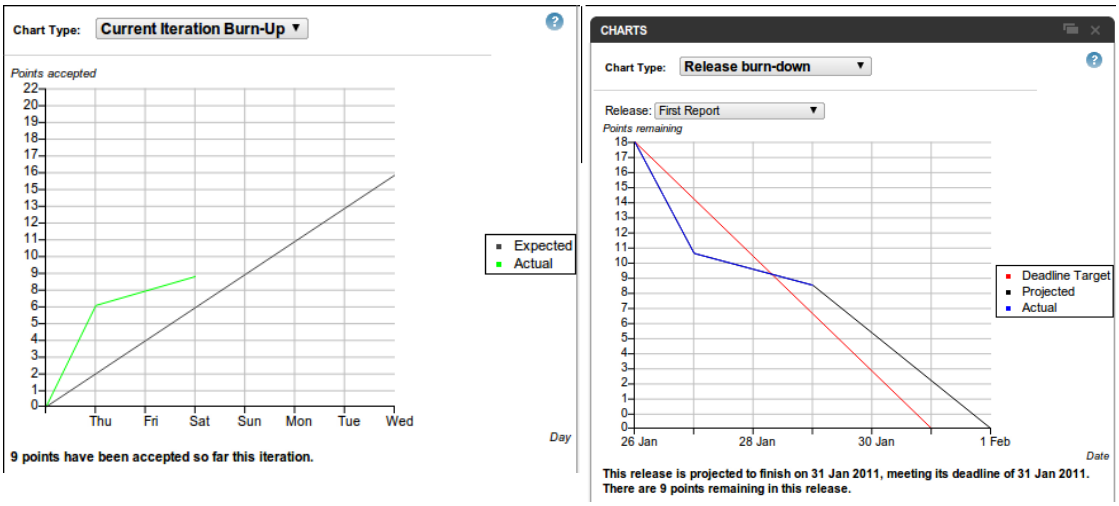
Project Management

We are managing this project using a modified SCRUM methodology centered around the Pivotal Tracker software tool. With this software we are able to collaboratively decide on tasks that will need completion in the short term, such as “Determine problem requirements”, mark what event or date they must be completed by (“First Report”), indicate who has started work on them and when, assign a number of “points” to the task and track the status of that task. The software then automatically fills “iterations” (sprints in the SCRUM terminology) with tasks to be completed, tracking how fast we progress through the points assigned to determine a project “velocity”. This can then be used to predict completion of events by a given time, as well as monitor where various tasks may be falling behind schedule. Tasks can also be assigned to a certain sub-team so each sub-team can see what tasks are set for them to complete.

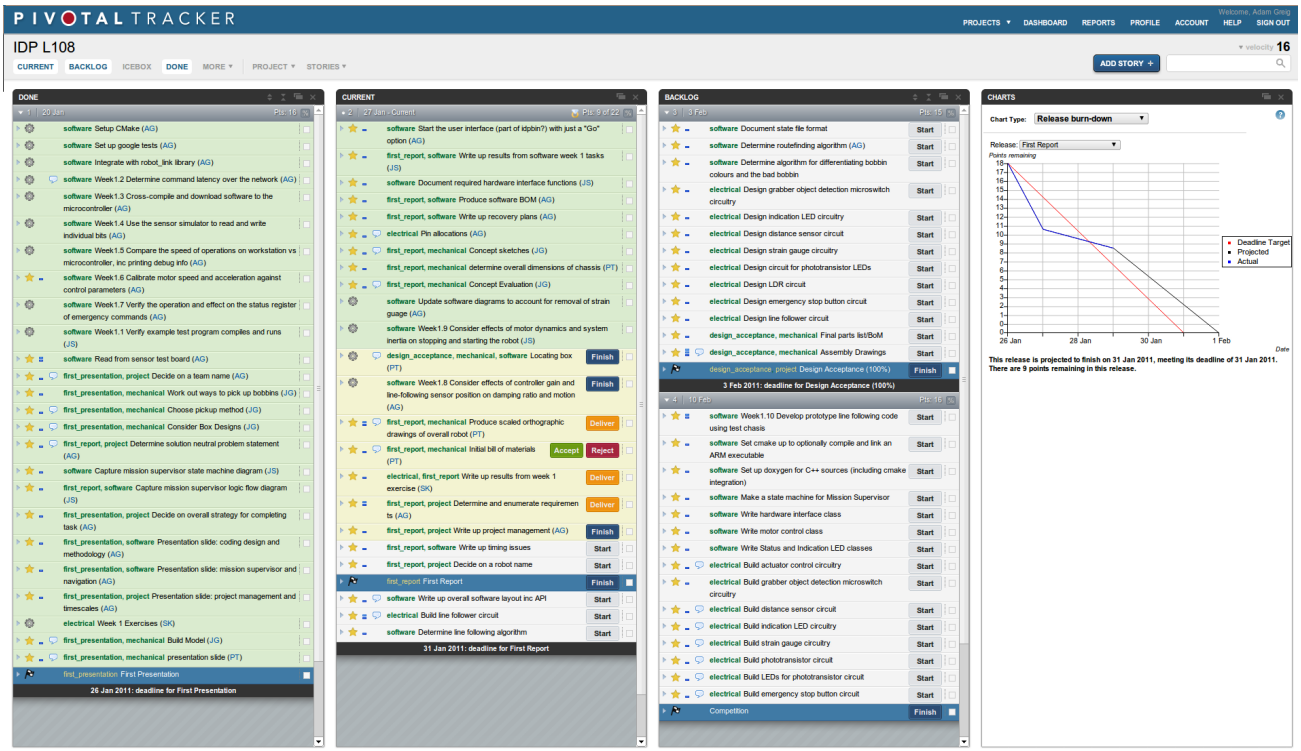
By populating the Tracker with all the tasks we can foresee needing to be completed and assigning a number of points approximately proportional to the time we expect that task to take, we are able to quickly check that we are on schedule for completion by the appropriate dates for each deadline, as well as identifying what might need more attention.

This does not provide any mechanism for managing dependencies between tasks but does allow tasks to be ordered in the “backlog”, so we are able to decide what needs to be worked on first when a priority ordering is important.

In the two charts below, the left shows the current points awarded in the current iteration. The green line is actual points, while the black line reflects the average that might be expected based on the previous iteration. We can use this chart to show how we’re doing compared to the previous week. The right hand chart shows the burn-down for the First Report release, with the red line indicating the rough rate points should be completed to finish on time, the blue line showing how many points have been awarded to date and the black line a prediction based on the current velocity of when we might finish. We can use this to quickly check how we are doing for a given release.



Below is a screenshot of the Pivotal Tracker interface, showing last week's tasks on the far left, the current iteration's tasks on the inner left (with some in a state ready to be delivered or accepted, and the already accepted tasks at the top), the backlog on the inner right (tasks that we will be starting in the next iteration or two) and the current release burn-down on the far right.



Software

Week One Tasks

The time to run a test instruction was determined to be approximately two milliseconds. This means we should be able to poll the line following sensors at approximately 500Hz, which is equivalent to approximately 0.5mm of distance travelled by the robot. This should be sufficient to detect junctions in plenty of time, and as such we intend to run the code on the workstation rather than the microcontroller for ease of user interaction and debugging. However, our build environment will also cross-compile the software such that it can be run on the microcontroller easily if this proves necessary.

The I²C I/O expander on the dummy sensor board has its address settable using three pins, controlled by switches, to determine the lower 3 bits of the I²C address. On our robot, we will manually set these by tying the pins to either ground or VCC, with different addresses for each chip.

Test code was written and compiled to run the motors on the dummy chassis, displaying an obvious need for tests in order to get the maximum acceleration (minimum ramp time) without causing wheel slip. We recorded the unloaded speed of the motors against the control parameter so that approximations of distance travelled can be made by the robot when needed once combined with knowledge of the wheel radius.

Emergency stop functionality was verified and some test code was written to handle an emergency stop condition, which will be carried across to the final robot. In an emergency stop condition, the robot resets itself to the starting position but retains its state, such that it knows what task to continue with.

When implementing the line following algorithm, we will have to decide on how we position the sensors and what feedback gain we employ, with consequences for the system's motion and damping ratio. A high gain allows us to react sufficiently to errors in position and prevent the line from being totally lost, but will increase the likelihood of the system going unstable. The gain parameter must therefore be carefully selected for. Some predictions of a suitable value can be made using systems theory, from which manual fine tuning may be required for optimum results. Our choice of sensor position will also have a strong effect on the system's behaviour. If the edge line sensors are a long distance from the line in the correct position, it will take a significant amount of position error before they pick the line up, resulting in a low damping and highly oscillatory motion of the system. If, on the other hand, the edge sensors are too close to the line, they are likely to pick it up very quickly on small position errors and result in a twitchy and possibly unstable line following motion.
A mechanical mounting that allows for the sensor position to be easily adjusted will make fine tuning this parameter possible.

The inertia of the robot will have a significant effect on the design of the software based control system. It is vital to ensure that wheel slip does not occur, as one wheel having traction and the other not could turn the robot so quickly that the line is lost. In addition, if the navigation system's position estimate has any basis in motor run-time, wheel slip will result in a serious deterioration of accuracy.

Ramping the motors should allow us to have the robot accelerate as quickly as possible without slipping.

If the robot powertrain has a high inertia, simply removing power to the motors may not stop the robot quickly enough. In this situation, there is a significant possibility that the robot may overrun a junction, and therefore have to stop and reverse to find it again, which is something that is best avoided. It may be that we have to brake actively in order to reduce stopping distance to an acceptable value.

Hardware Interface Functions

Software is required to communicate with most pieces of hardware on the robot.

The LEDs for status and indication, as well as those for the colour sensing unit, will be controlled directly from the micro via an I²C I/O expander. Two digital outputs are also required for the two actuators, which are to be driven by transistors controlled from the micro. Three analogue inputs will be required, which are sampled by the ADC on the PIC and read into the main program via library functions. These are the two LDRs and the IR distance sensor. The signals from the IR detectors used for line following will be passed to the micro through the I²C I/O expander via Schmitt trigger ICs.

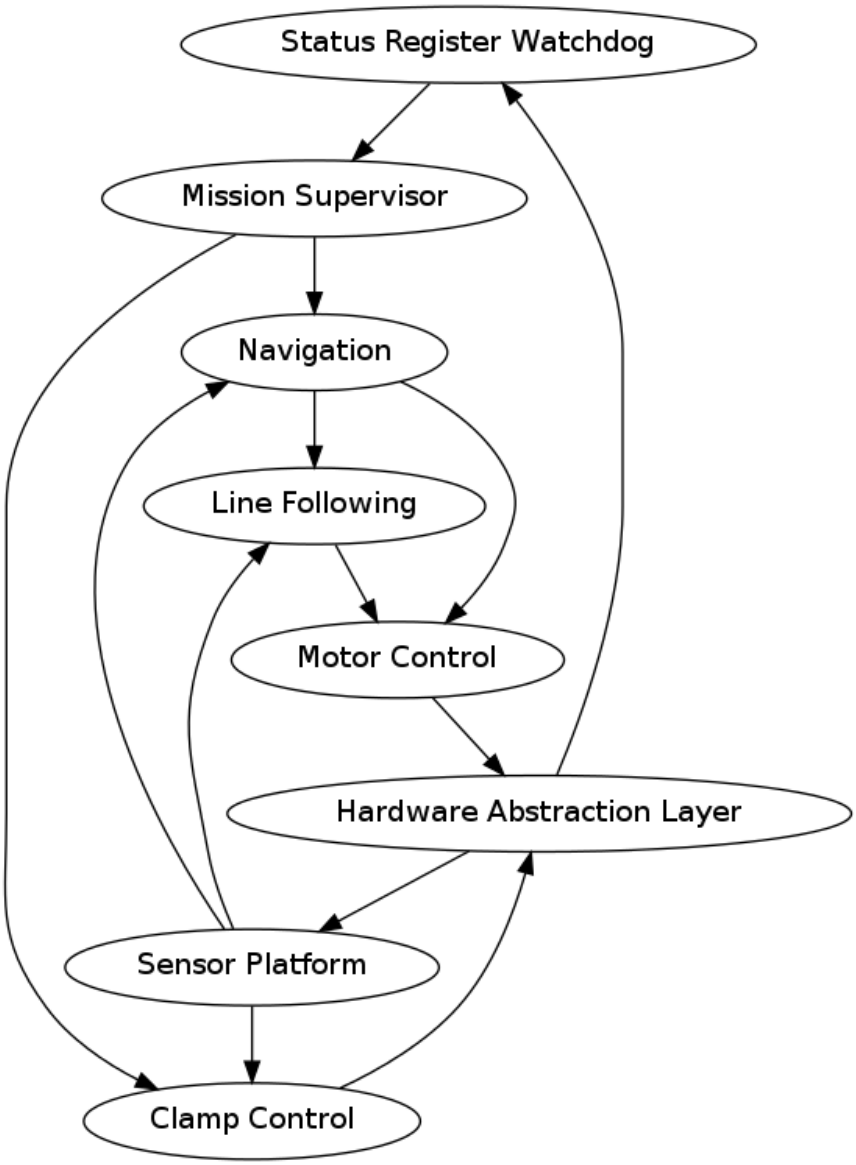
Functionality to interface with this hardware will be encapsulated within the hardware abstraction layer module, so that the rest of the software does not need to worry about robot-specific hardware or communication issues. This helps ensure modularity of the produced code and makes unit testing easier and more reliable.

The status register is read by the mission supervisor to detect failure conditions such as an I²C bus error or the emergency stop condition so it can take the appropriate recovery action.

Code Compilation and Build Environment

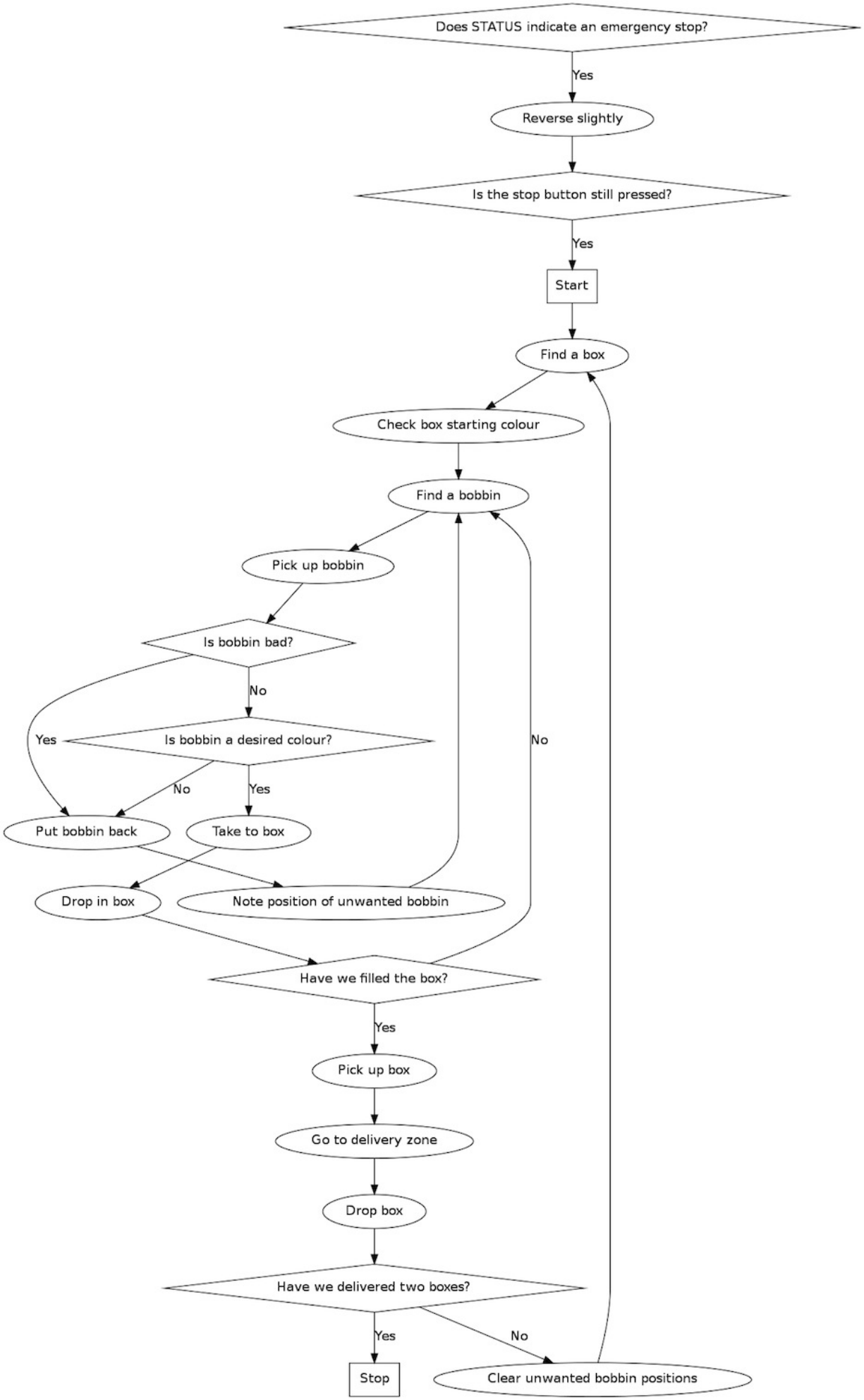
The build environment has been set up using cmake for fast and automated compilation and test running and code is kept under version control using the Git SCM. This allows us to develop the software easily from any location without the requirement of having a robot or access to the teaching system, and ensures that mistakes are caught quickly as changes to the code can be easily reviewed and the unit tests are run on every compilation.
The source code is commented throughout using the format accepted by Doxygen, so automated HTML documentation can be produced detailing the software API and functionality. This makes it easier to quickly see what interfaces other classes provide without having to check their actual source code. The documentation is also generated automatically by the build process.

Code Layout

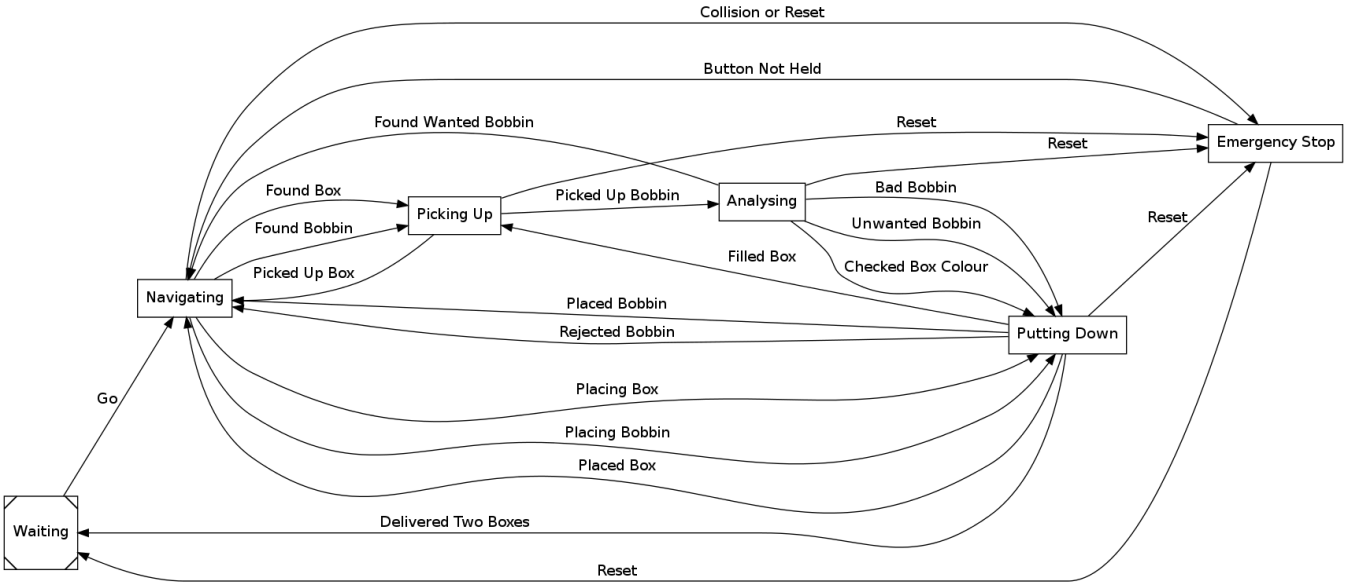


Mission Supervisor

The mission supervisor will be written such that it contains no control code and therefore is pure logic. The logic flow diagram is shown below:


Does STATUS indicate an emergency stop?
Yes
Reverse slightly
Is the stop button still pressed?
Yes
Start
Find a box
Check box starting colour
Find a bobbin
Pick up bobbin
Is bobbin bad?
No
Yes
Is bobbin a desired colour?
No
Yes
No
Put bobbin back
Take to box
Drop in box
Note position of unwanted bobbin
Have we filled the box?
Yes
Pick up box
Go to delivery zone
Drop box
Have we delivered two boxes?
Yes
No
Stop
Clear unwanted bobbin positions

Additionally, it will be designed as a state machine whose states and transitions are shown in the following state diagram:



Recovery Strategies

There are several failure conditions which may be run into during the task. A list and the appropriate recovery strategies are given below.

Lost position

In this state, the navigation subsystem should have a position estimate but knows it is not accurate. This could be due to not seeing a line when one should exist, or after having come from an emergency stop condition. Navigation should then direct the robot to where the nearest line should be according to its estimate, and once it finds a line, follow it whilst looking for landmarks that will allow a firm position estimate to be obtained. The robot can then continue the task.

Dropping a box or bobbin

The grabber mechanism will be able to detect whether it is holding something, but since friction is used to keep the item in the grabber claws, there is some chance the object could be dropped. The reaction will depend on the object, the position on the board, and how far the object will have fallen. Dropped bobbins from a height are likely to be a lost cause and will probably be ignored, whereas dropped boxes or bobbins dropped from a lesser height can probably be recovered.

Failed attempt to pick up an item

In this case, it is a simple case of determining that the item has not been successfully picked up, and making another attempt.

Emergency stop

Due to our positioning of the emergency stop switch means that an emergency stop condition means the robot has collided front on with something. In this state, we stop the motors immediately and reverse gently for a moment to release the switch. A collision will only have occurred if navigation's position estimate is significantly wrong, so we then jump to the "Lost" failure condition above.

Cannot determine bobbin colour

Calibration and testing should mean that we can reliably detect bobbin colour. However, if after several attempts, the colour of a bobbin is ambiguous, then replace the bobbin and move onto the next one.

Unexpected hardware or logic failure

In this state, the best course of action is almost certainly to navigate back to the starting zone and attempt to continue the task from there. In order to avoid undetected hardware failures, inbuilt self-test features will allow us to verify the functionality of all the hardware and software before the task is started.

Timing Issues

We anticipate that the primary timing concern will be the latency of network commands when the robot is driving and attempting to follow a line, but there is a time delay between reading a sensor and controlling the actuator. As a system, this adds a delay factor that will make it harder for the system to be stable and well behaved.

However, the tests we conducted did not reveal significant cause for alarm. The code can be run on the embedded microcontroller, in which case network communication delay no longer has an effect, but the CPU speed of the embedded microcontroller might be an issue. Further tests showed that it is unlikely this will cause significant problems, as the embedded CPU operates at hundreds of MHz and our compiled C++ code should be able to perform the task with plenty of spare CPU resources.

Initial Bill of Materials (Software)

Code modules:

Hardware Abstraction Layer, 200loc
Status Register Watchdog, 100loc
Mission Supervisor, 600loc
Navigation Control, 600loc
Line Following, 300loc
Motor Control, 200loc
Sensor Control, 300loc
Clamp Control, 300loc
User Interface, 300loc

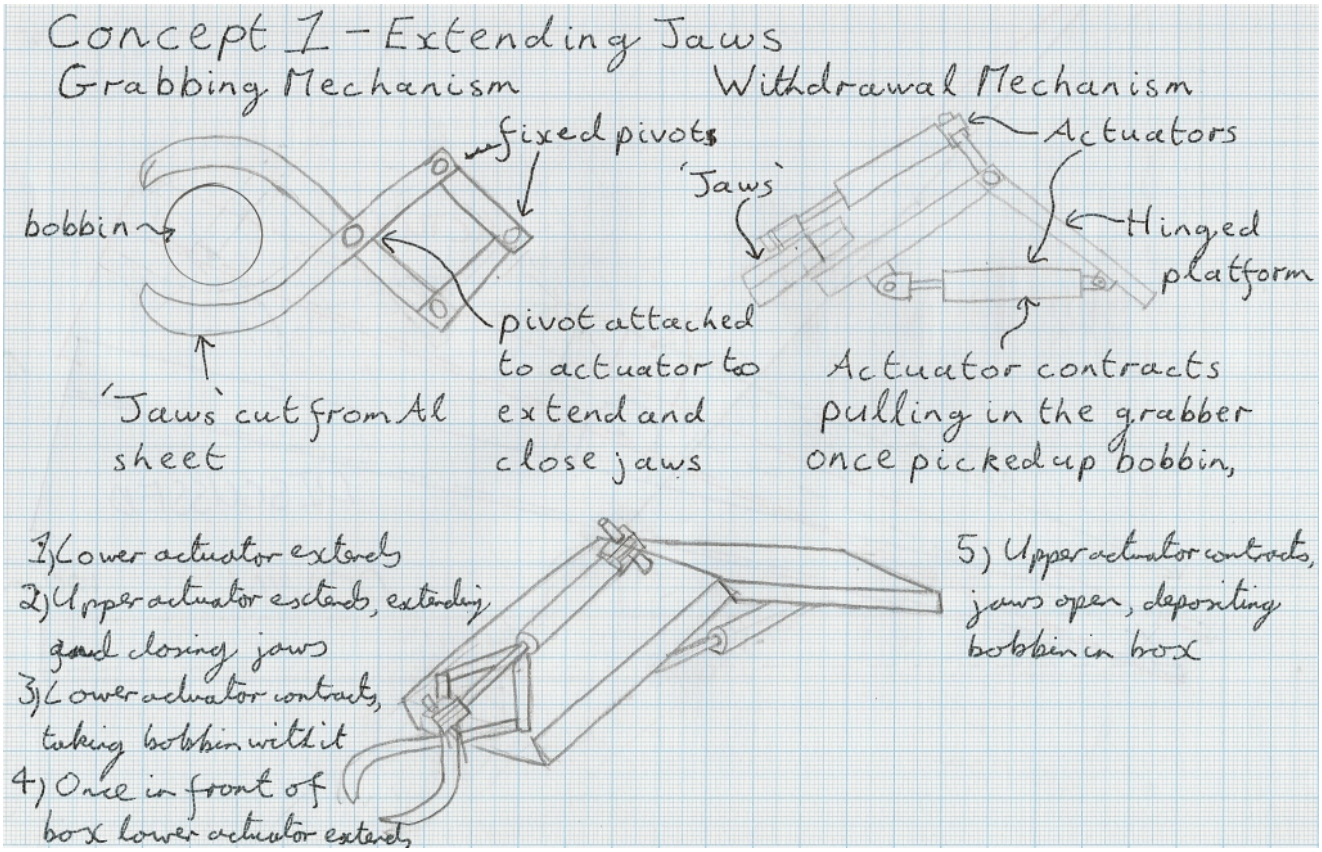
Total line count estimate: 2900 lines of code, not including unit tests

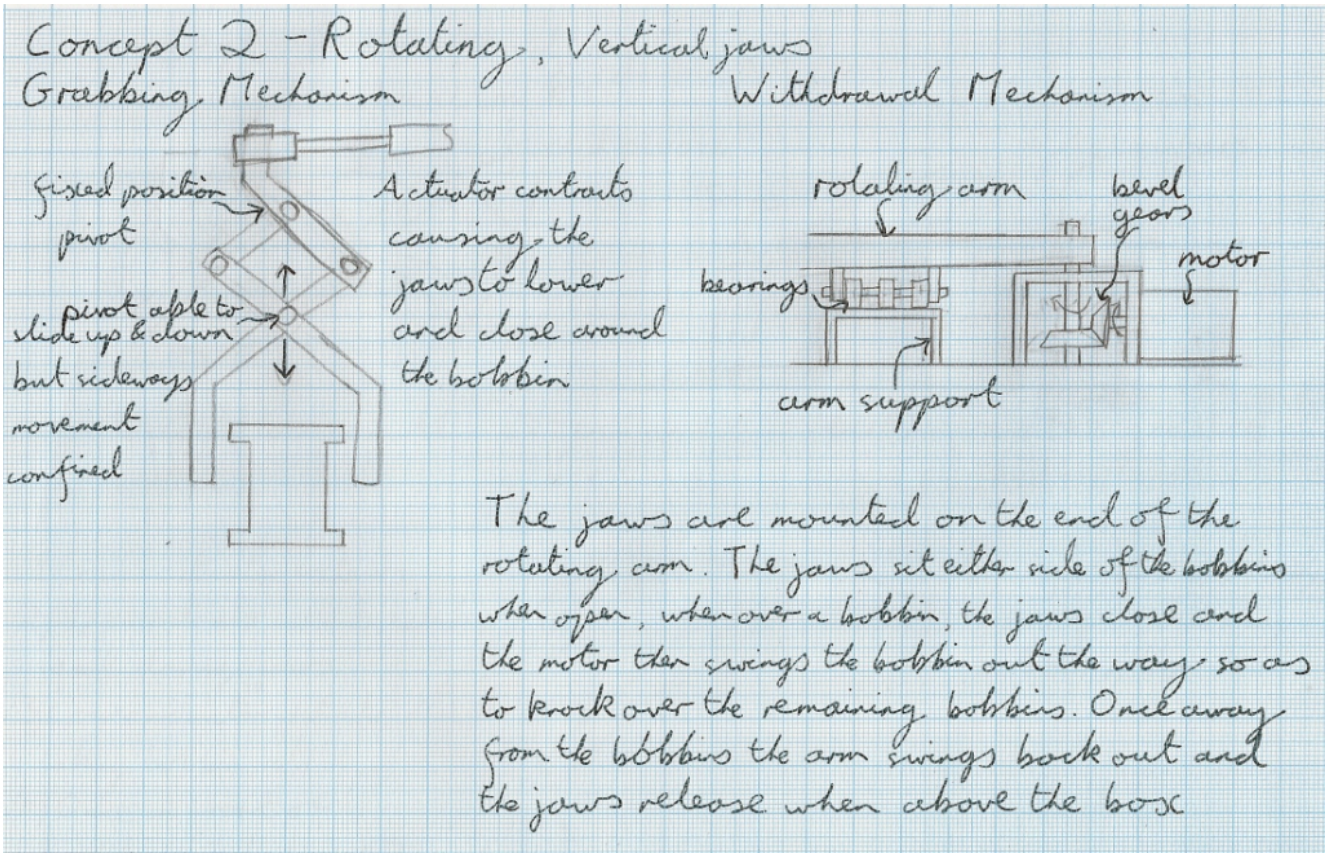
Software-Hardware Interface

Hardware	Pin Type & Count	Port and Pin Allocation
Line following sensors	4 digital inputs	Port 000 P0-3
Microswitches	2 digital inputs	Port 000 P4-5
Indication LEDs	3 digital outputs	Port 111 P0-2
Color sensing LEDs	2 digital outputs	Port 111 P3-4
Bad bobbin sensing LEDs	1 digital output	Port 111 P5
Actuator Control	2 digital outputs	Port 111 P6-7
Distance Sensor	1 analogue input	ADC0
Colour sensing LDR	1 analogue input	ADC1
Bad bobbin sensing LDR	1 analogue input	ADC2

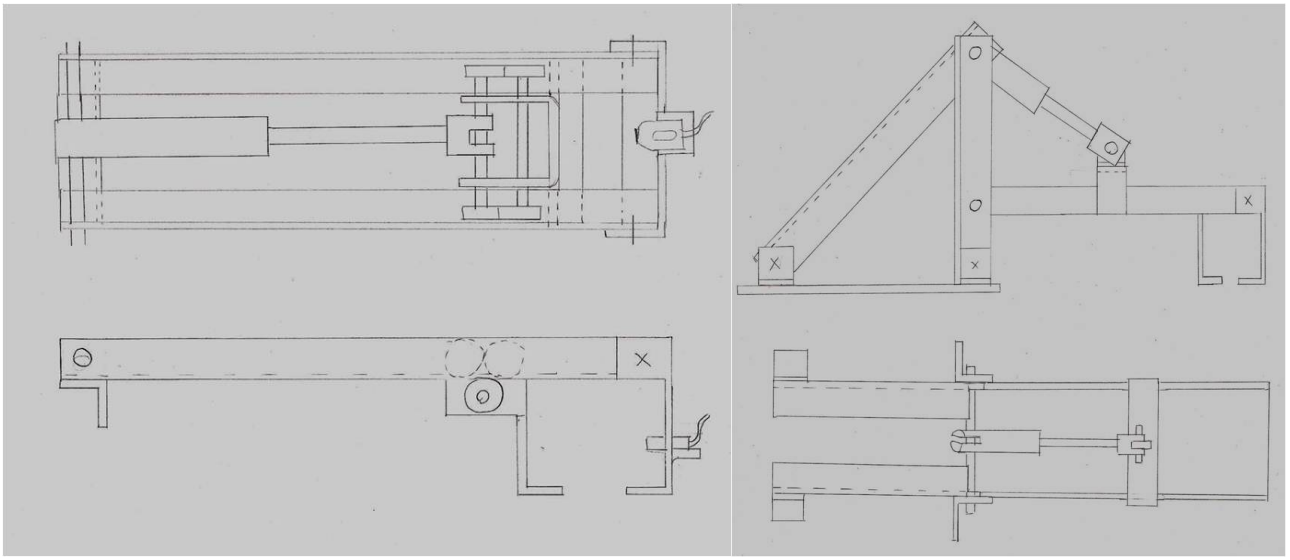
Mechanics

Concept Sketches





Concept 3



The grabber protrudes over side of chassis with a fixed plate that sits behind the row of bobbins and a moving plate on the opposite side to pick them up once correctly positioned. This is done using an actuator connected to the moving plate that extends once in position to grab the bobbin. Bearings above and below the flange of the aluminium angle prevent vertical movement and rotation of the moving plate.

The grabber is then raised with the other actuator such that it does not knock over other bobbins when moving to the box. Once the robot has turned around the arm lowers and when above the box the first actuator pulls the plate back allowing the bobbin to drop into the box below.

Concept Evaluation

Concept 1

The advantages of this design are that it is the only design that has the ability to push the bad bobbin off the back of the pickup area, by extending the jaws while the lower actuator is contracted, and then extending the lower actuator again.
A disadvantage of this design is that when the jaws are withdrawn they rotate and also lower, this could result in the robot lifting itself up on the side closest to the pickup area, putting lots of strain on the jaws through which this force is transferred.

Concept 2

A disadvantage of this design is that when rotating the arm there is a high probability of knocking over other bobbins, while this will have no effect on the competition run as they will be replaced when the robot is far enough away, in a real life situation however this would be undesirable due to possible damage to the products being moved and would also require a person to follow it round and replace then, involving more effort than them doing it in the first place

Concept 3

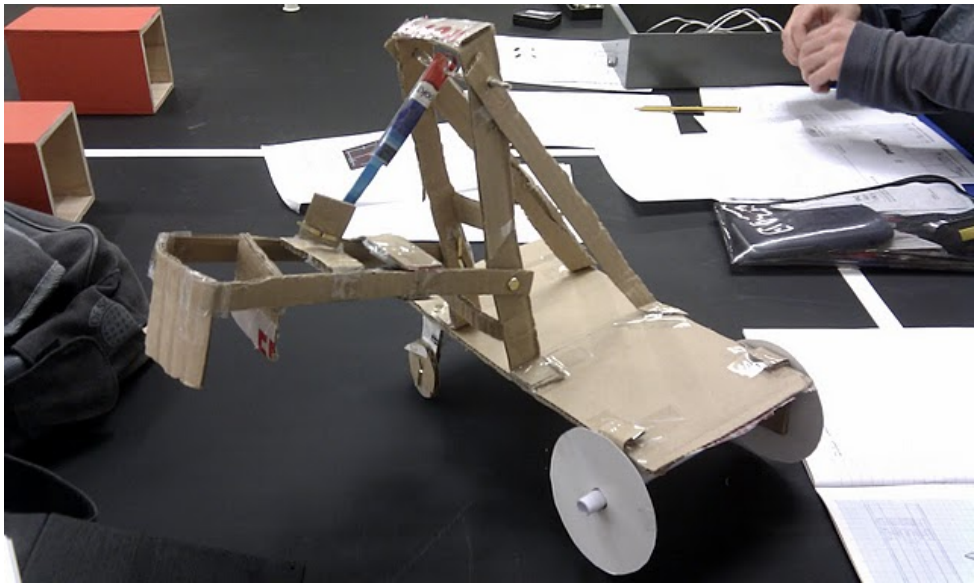
An advantage of this design is its simplicity over the other two, the grabber mechanism, is a plate, supported by bearings, which is pushed by an actuator, the withdrawal method is another actuator attached between the top of the A-frame and the grabber. This design will mean that no other bobbins are knocked over while removing the bobbin.
A disadvantage of this design is that it is unable to push the bad bobbin off the back of the pickup area.

	Concept 1	Concept 2	Concept 3
Simplicity of grabber mechanism	0	0	+1
Simplicity of withdrawal mechanism	0	0	0
Risk of knocking over other bobbins	0	-1	0
Ability to push bad bobbin off pickup area	0	-1	-1
Strength of grabber (to grab the bobbin)	0	0	0
Strength of grabber (to other forces likely to be applied during use)	0	+1	+1
Total	**0**	**-1**	**+1**

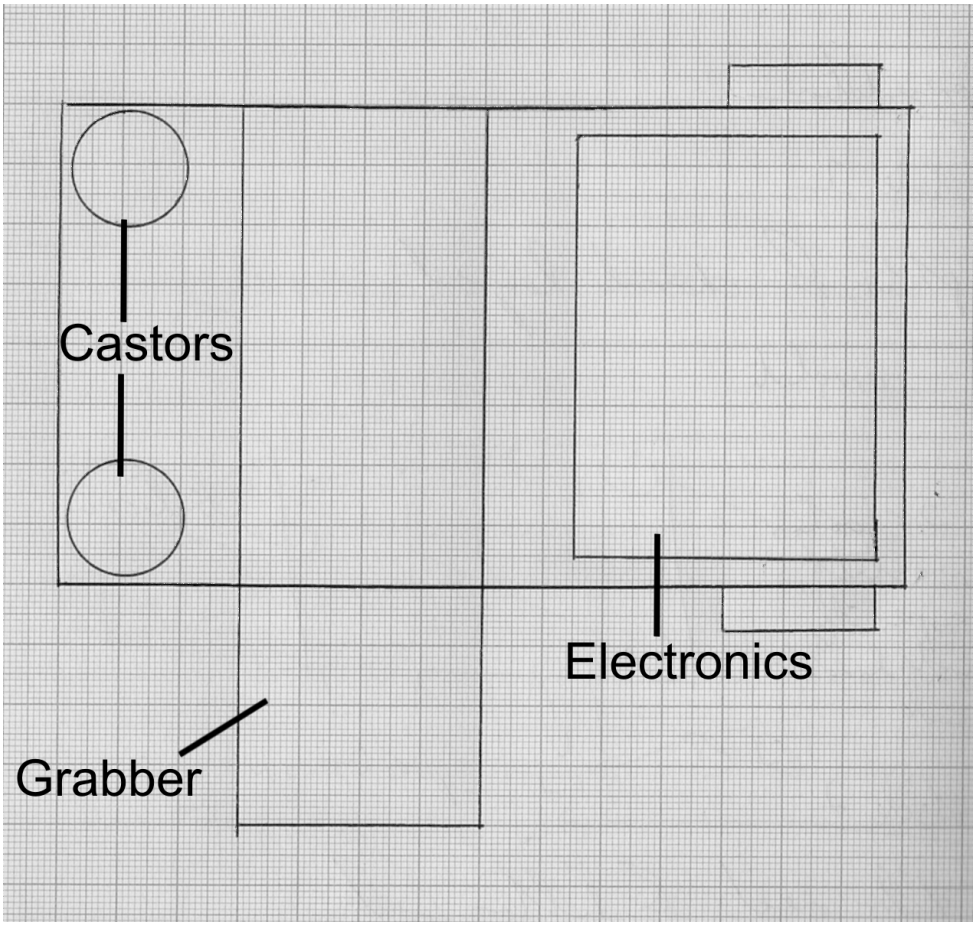
Based on the evaluation table and the advantages and disadvantages of each concept, concept 3 was chosen, it's main disadvantage is that it can't push the bobbins off the back of the pickup area, however it is felt that this is only a small issue since 6 points are still gained for leaving the bobbin where it was (only an additional 2 are gained for knocking it off.

Model & Tests

Having chosen the design a model was made to allow for easier visualisation of the concept as well as the necessary dimensions to allow the robot to function as intended.



Tests were also carried out to determine the orientation and layout of the chassis. These were done with the dummy chassis to observe the behaviour when run as front and rear wheel drive as well as to find what motor/wheel combination should be used. For stability we chose to have a wide chassis, due to the grabber mechanism needing to hold a box we felt that there was a strong chance that it would be unable to go through the restriction to reach the deposit area, as such we added load to the chassis and tested it going over the ramp. It was found that the best combination to achieve this was the large motors with large wheels, driven by the rear wheels. From These decisions a chassis layout was decided.



Initial Bill of Materials (Mechanical)

- 2 x Wheels, soft rubber tyre, 100 dia
- 2 x Castors, soft rubber tyre, 50 dia
- 2 x Large motor/gearbox 12v DC, 20rpm/40 rpm
- 2 x Pneumatic actuator 10 bore x 45 stroke
- Pneumatic valve assembly
- Pneumatic hose 1200mm
- Mild Steel sheet 23swg, 400mm x 405mm
- Aluminium Sheet, 1.5mm, 100mm x 165mm
- 4 ball bearings

Aluminium angle 530mm

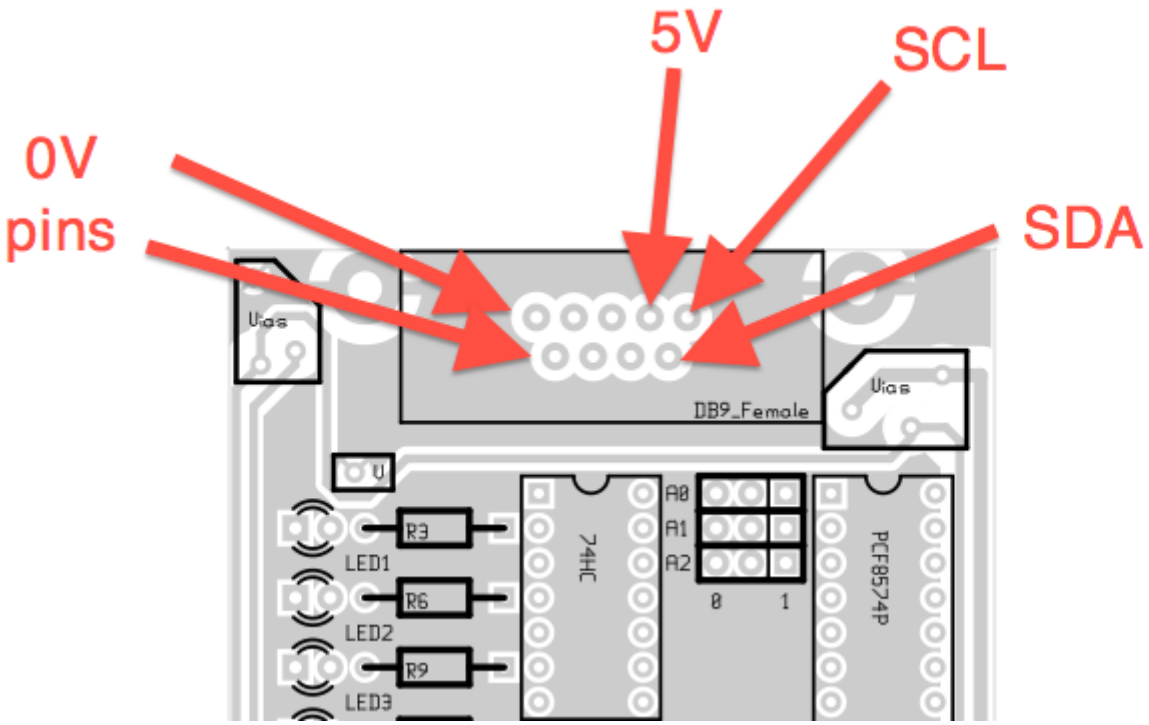
Electronics

Week One Exercises

During the first week the electronic team worked through the exercises in a methodical process and the results are displayed below.

Exercise 1

The electrical connections to PCB 1 were checked and below is an image with the 0V, 5V, SDA and SCL pins labelled.

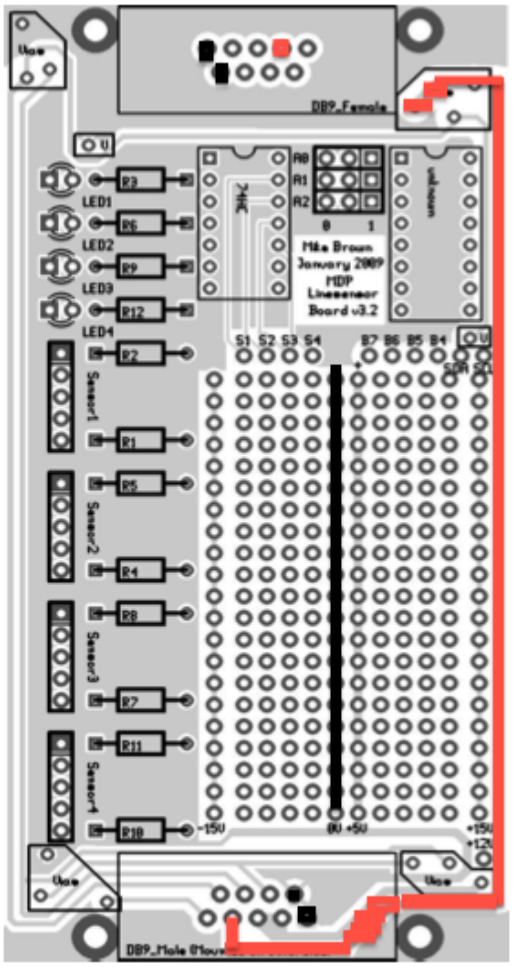


Exercise 2

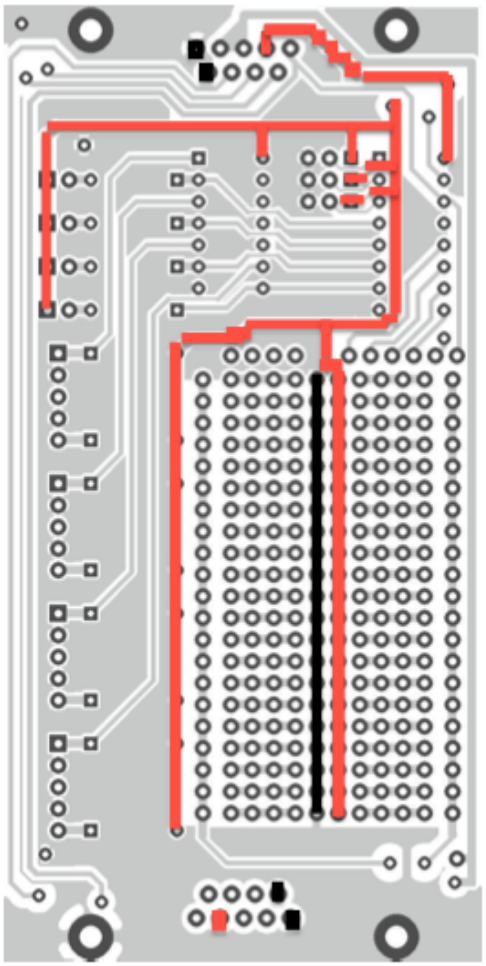
Using the D-type connectors the continuity between the 0V, 5v, SDA and SCL pins was confirmed.

Exercise 3

In the diagram below the 0V and 5V tracks on PCB1 are displayed as black and red lines respectively.

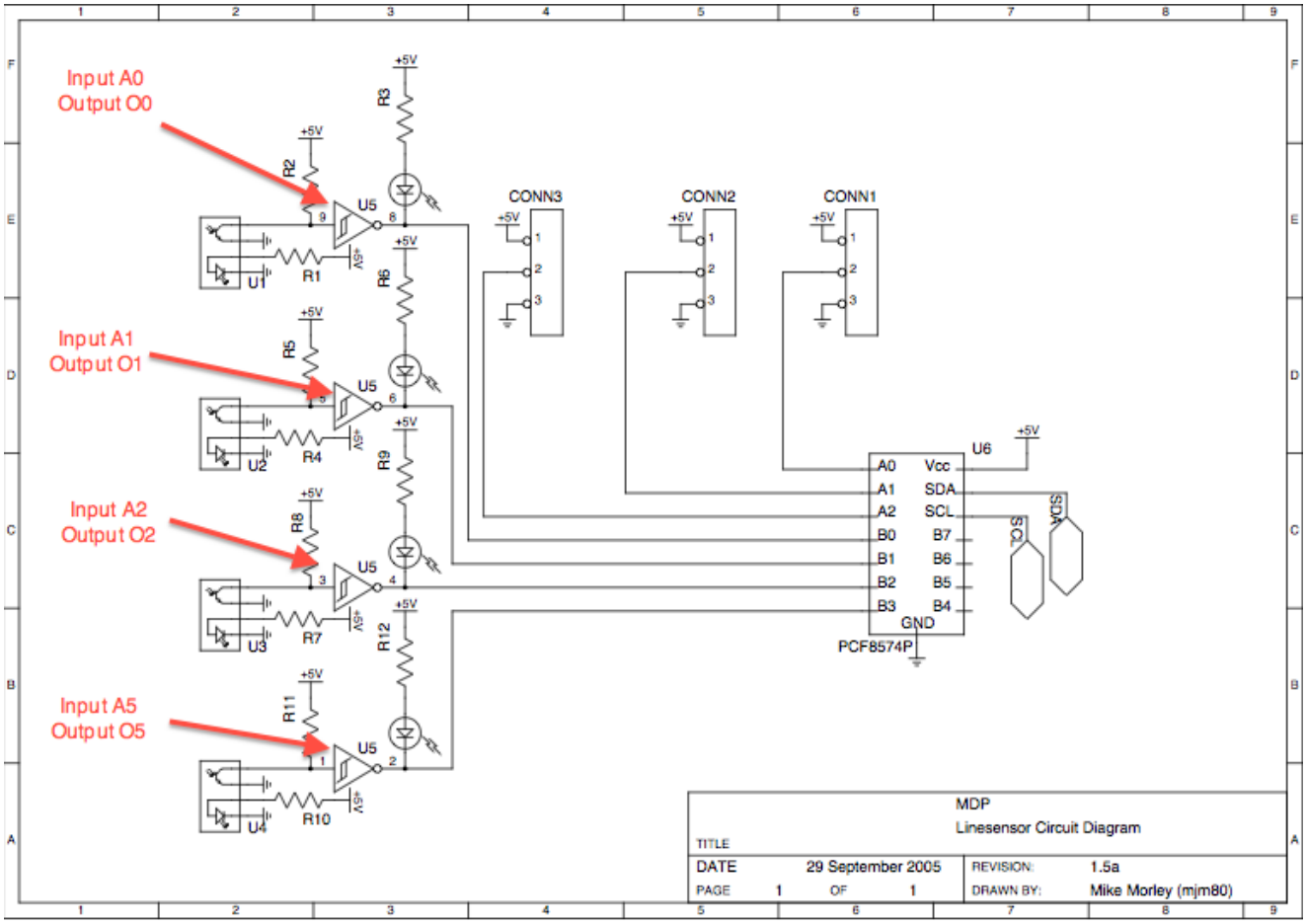


Linefollowing (component side)

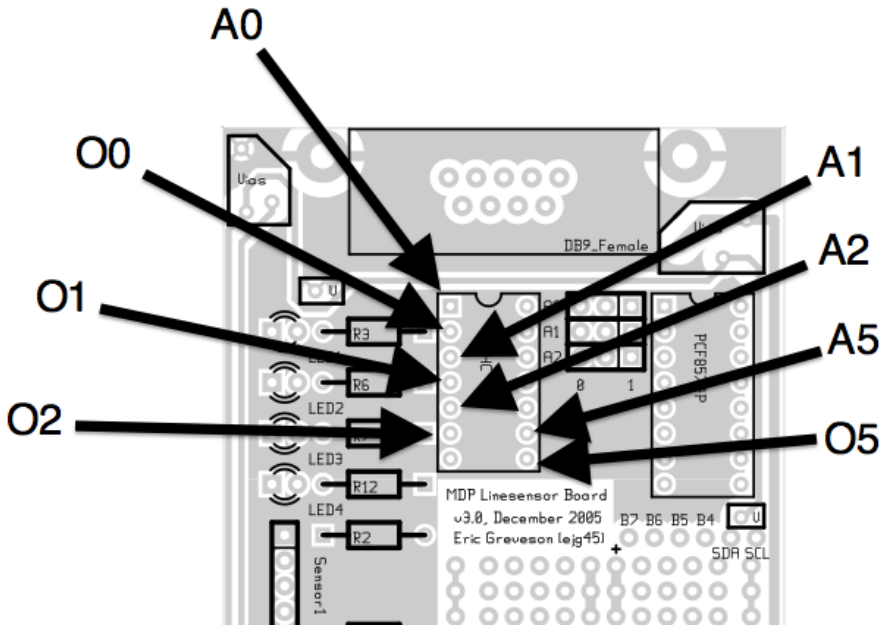


Linefollowing (solder side)

From the image below the input and output pin numbers of the Hex Schmitt inverter chip corresponding to each of the line follower circuits are displayed.

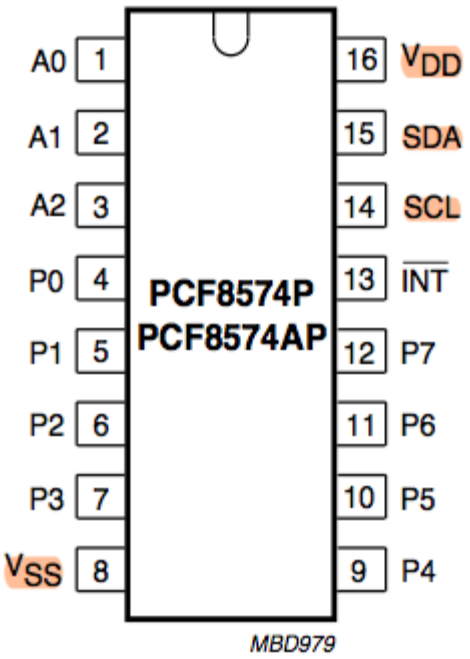


The following image shows where the previously mentioned input and output pins for the line follower circuits are found on PCB 1.

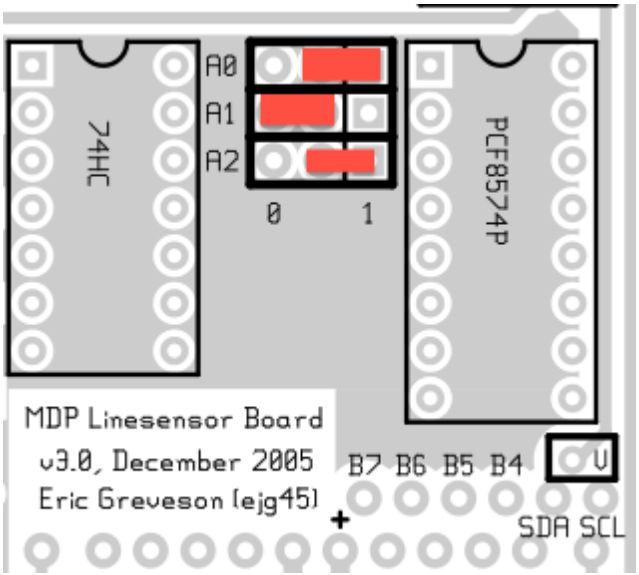


Exercise 4

The diagram below shows the pins on the PCF8574P chip, with the 0V, 5V, SDA and SCL pins highlighted (V_{ss} and V_{DD} correspond to 0V and 5V respectively).



By observing the underside of PCB1 it was determined that the first three pins of the PCF8574P chip are connected to the central column of the 3 by 3 grid to the left of the PCF8574P chip. These pins are used to set the address of the PCF8574P chip in order for the microcontroller to determine which chip it is interfacing with. The outside columns of the 3 by 3 grid are connected to the 0V and 5V tracks. The jumper pins are then used to connect the central pin to one of the outside tracks depending on whether you want the pin to have a high or low value. By having a combination of 3 jumpers it is then possible to set a unique sequence of high and low voltages for the address pins of each chip. The image below shows how you would use the jumpers to set an address of 101.



By tracing the paths on the underside of PCB1 the pins corresponding to each of the line following sensors are:

Sensor 1 = P0 (pin 4)

Sensor 2 = P1 (pin 5)
Sensor 3 = P2 (pin 6)
Sensor 4 = P3 (pin 7)

Notes on electrical circuits:
1) The SCL and SDA lines are used to connect the PCF8574 chips together.
2) The jumpers will be set up for the two microchips to have an address value of 000 and 111, for simplicity and not mixing the 0's and 1's
3) 4 IR sensors will be used for the line follower, two above the white line for better accuracy, one left and one right for detecting the black and the 90 degree turns.
4) A green LED and a red LED will be used to identify the colour of the bobbins, which is more accurate than the white LED that comes with the LDR, since bobbins are white, green and red.

Required Circuits

In order for the robot to be able to carry out the tasks required of it, numerous circuits are needed. The required circuits are listed below.

- 4 line follower circuits
- 2 actuator circuits for controlling the grabbing mechanism
- 1 micro-switch circuit to determine if a box has been reached
- 1 distance sensor for detecting a bobbin
- 1 micro-switch circuit for general collision detection
- 1 push button for reset
- 1 LDR circuit for determining the colour of a bobbin
- 1 LDR circuit for detecting the bad bobbin
- 3 LED circuits will be used to indicate the state of the robot
- 3 LED circuits will be used in conjunction with the LDR circuits for detecting the bad bobbin and colour of bobbins

Trade off between using hardware/software

When designing the circuits to be used in the robot we had to account for the advantages and disadvantages of using software over hardware. For example logic function can be defined in software or through hard-wired logic. By using software over hardware there are speed advantages in the running of the program as well as other advantages such as minimising the number of pins used. However there are also drawbacks such as minimising the number of pins used. However there are also drawbacks such as more complex circuitry being used and more space being used on the PCBs. There can also be a loss of information by relying on the hardware to determine certain features of the robot's environment. Generally we have chosen software over hardware in order to maximise the accuracy of our sensors. For example we have chosen to use analogue to digital ports to determine the colour of the bobbin instead of using hard-wired logic. Although this will make the coding of the software more difficult it means that the threshold values for determining the colours can be easily edited at a later time to improve accuracy when testing the whole robot.

Initial Bill of Materials (Electrical)

- 2x PCBs
- 2x PCF8574P microchips
- 4x IR Sensors
- 2x LDRs
- 10x LEDs
- 2x Mosfet transistors
- 1x Distance sensor
- 27x fixed value resistors
- 1x 74HC14N - Hex scmitt-trigger inverter
- 2x microswitches