RoboCup Junior Rescue Line 2023

Engineering Journal

CatBot NEO

*This version of CatBot NEO's Engineering Journal is up to date as of **Monday, 11th of July 2023***
All code, as well as other technical documents (Poster & TDP), can be found on our GitHub Repository

Team Information

The "CatBot NEO" team, formerly known as "CatBot", was established in Melbourne, Australia in 2016 as a part of the beginning of St Michael's Grammar School's Robotics Club. The robot has since evolved over the years, having started with Lego Mindstorms, building up through Arduino, and finally to a Raspberry Pi based robot.

The team consists of two members, Zac McWilliam, and Luna Verratti. Luna primarily handles the software side of the robot, and Zac contributes to both the hardware and software.

The team has competed in approximately 11 events in Australia, key results include: *(no access to data prior to 2021)*

- 2016 Melbourne Regional and Victoria State event
- 2017 Melbourne Regional and Victoria State event
- 2018 Melbourne Regional and Victoria State event
- 2019 Melbourne Regional event
- 2019 Victoria State event
- 2019 Australia National event *(Melbourne)*
- 2021 Australia National event *(Online)* **2nd Place** (Scoresheet Link)
- 2022 Melbourne Regional event **3rd Place** (Scoresheet Link)
- 2022 Victoria State event **1st Place** (Scoresheet Link)
- 2022 Australia National event *(Adelaide)* **2nd Place** (Scoresheet Link)

Members regularly meet in person and over online platforms such as Discord to work on the robot, this can be as often as daily when schedules align correctly, this has significantly increased in frequency during 2023.

GitHub is used as the primary source of version control for working on the robot, however, as code is stored on the Raspberry Pi itself, we use GitHub more as a backup and record keeping device, pushed to directly from the robot after changes have been made.

Documents such as the Engineering Journal are shared over OneDrive so that both team members can continuously update it during the design and development process.

CatBot NEO - 2016-2023

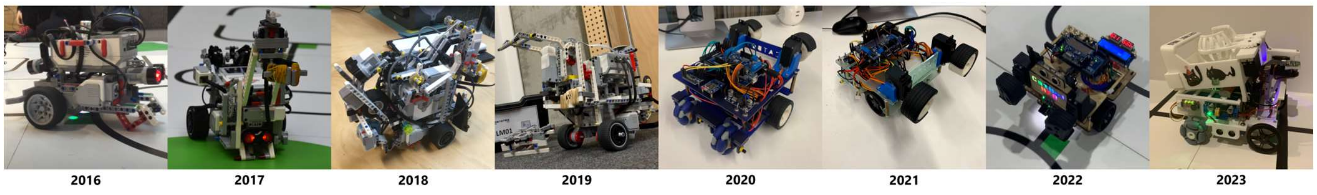


Table of Contents – Page 1

Table of Contents – Page 2

We have decided on a summary-based structure for our Engineering Journal, this will hopefully provide more clarity when reading through each section as matching information can be put together easily where relevant.

Logs will be in date ranges, from a single day, or to at most a week (labelled accordingly at the start of each entry), depending on how many significant changes occur and what makes the most sense to compile together.
We hope this way, logs will be much more readable by others, compared to being messy and spontaneous.

All journal entries are contributed to by both Zac and Luna, then finally reviewed and formatted by Zac

April

Week of 3/4/2023

Tasks Completed

- Started to investigate RCJI rules, and how they differ to RCJA.
  - Completely new evacuation zone
  - Need to pick up balls instead of cans
  - New intersection marker logic
  - Stop program on red line
  - Steep ramps, 25deg instead of 17deg

Comments

- Will likely attempt a level 2 evacuation, and a level 1 blue cube rescue, unless time permits.
- Need a new mechanism to pick up balls, perhaps a claw in a shape to surround the 5cm ball?
- The steep ramps will force a new design, likely 4WD, we already had problems with 17° last time

Tasks Completed

- Looking back at CatBot NEO from 2022, what flaws were there? What went well?
  - Struggled to get up ramps
  - Fixed line sensor mount was unreliable
  - Arduino can be slow, and limits capability
  - Neat wiring and good planning helped make the robot very stable

Comments

- Ramps could be improved with 4 powered wheels, rather than just two.
- Use a sensor mounted on a hinge
- Raspberry Pi? This would also allow use of a camera for the evacuation zone.

Tasks Completed

- Ordered a 3D Printer (Bambu Lab X1C)

- Set up Fusion 360 team
- Created shared folder in OneDrive for files such as the Engineering Journal

- **Table 1 -** Created Project Plan

April	**May**	**June**	**July**
Investigate the new competition List the requirements Estimate cost and order all parts and tools Design CAD model	Assemble first prototype Create test code for each subsystem Mostly complete line following Basic obstacle avoidance	Implement intersections/green turns Evacuation Zone Finish off and optimize code Lots of testing and fixing	Travel to Bordeaux Any finishing touches right before the comp

Week of 10/4/2023

Tasks Completed	**Comments**
- Created a component list based off the old CatBot, as well as implementing new ideas.	- Creating this list prior to the creation of a 3D model or circuit diagram allows us to plan ahead and arrange parts in a very effective layout

New Considerations and Major Changes

▪ Use of a Raspberry Pi 4B ○ New motor controller (link) ○ Touch screen for debugging (link)	This will allow us to utilize a Raspberry Pi Camera V2 in order to effectively complete the Evac Zone. Optimally, using this camera to line follow would likely result in a much better outcome, however, due to a lack of time and experience, we have chosen to stick with a colour sensor/reflectivity array design.
▪ New colour sensors (Option A), or (Option B) ▪ Longer reflectivity array (QTR-MD-13A)	Our old colour sensors have been discontinued, option A appears most promising, but we will order both just in case to save time. The longer reflectivity array will provide more visibility of the line.
▪ Analog to i2C board (ADC Pi)	The RPi has no analog ports, a fix for this is the ADCPi which has 8 analog inputs and can be read via i2c.
▪ 380:1 Micro Metal Gearmotors (link) ○ Slim wheels, same as old CatBot (link) ○ Wide wheels, more traction? (link) ○ Omni Wheels, any direction (link)	These have more torque compared to the previous version of CatBot, this should help us get up the steeper ramps. I have included 2 options for wheels as we might need thicker ones, to be determined.

- Tested assembly of components in CAD using Fusion 360 (Fig. 1 and Fig. 2) - Parts ordered after successful mockup model	Before ordering, we checked that all the parts would generally fit together, this was confirmed, so parts were ordered. We used RCJA CatBot's design to base initial dimensions from (blue outline in Fig 1)

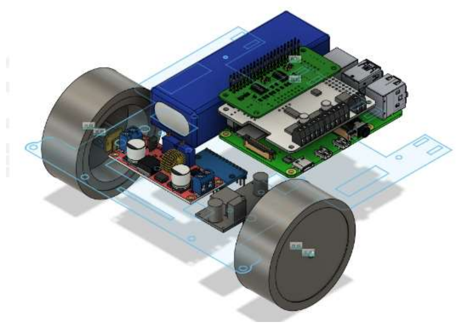


Figure 1 – *Initial mock-up using old base dimensions*

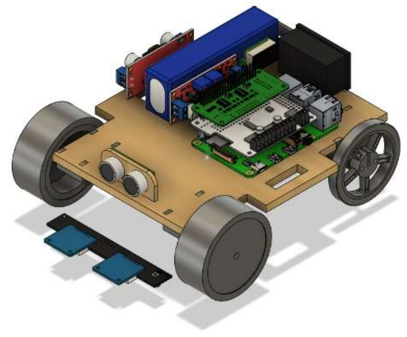
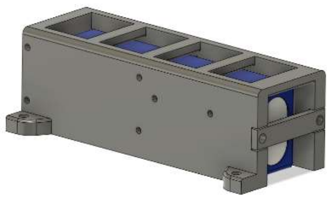
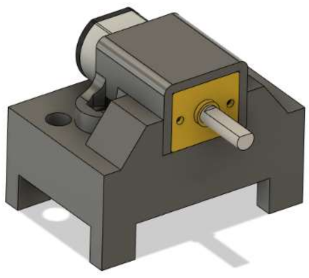
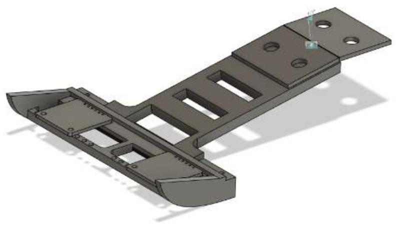
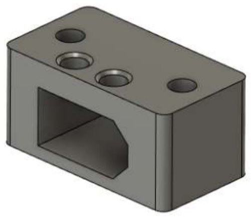
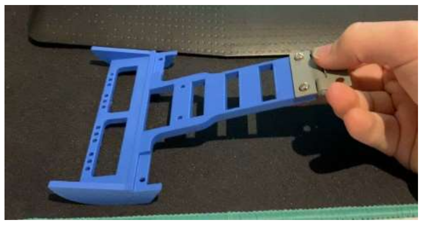


Figure 2 – *Rest of components mock-up, started designing a new base plate shape*

17/4/2023 to 19/4/2023

Tasks Completed	**Comments**
- Received 3D Printer, ran some test prints.	Very successful, will likely use for the robot base
- Created LiPo Holder (Fig. 3) ○ **Issue**: Heat insets didn't fit the holes **Fix**: Ensure hole sizes are 3.8mm	In the previous CatBot, taking the LiPo out was not possible. This will allow us to hot-swap the battery and always have one charging.
- Created motor holder bracket (Fig. 4)	As we are uncertain of the wheel size, this bracket can be adjusted in height and easily swapped out when needed.
- Created floating sensor mount (Fig. 5)	Uses a hinge, will buy from bunnings. Then need to adjust design to match the dimensions.
- Created XT60 battery connector holder (Fig. 6)	Provides a solid point of connection for the battery
- 3D printed all the parts designed so far	
- Tested floating sensor mount on the new hinge (Link) – Fig. 7 ○ **Issue:** It has a slight wobble side-to-side **Fix:** None required, acceptable for now	

 ***Figure 3*** *– LiPo Holder*	 ***Figure 4*** *– Motor holder bracket*	 ***Figure 5*** *– Floating sensor mount*
 ***Figure 6*** *– XT60 connector holder*	 ***Figure 7*** *– Sensor mount test*	

20/4/2023 to 23/4/2023

Tasks Completed

- Created an initial mock-up of the claws (Fig. 8).
- Printed test gears (Fig. 9) before printing the full assembly to ensure they work

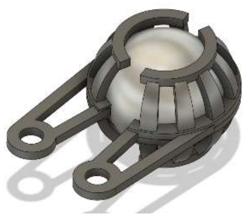
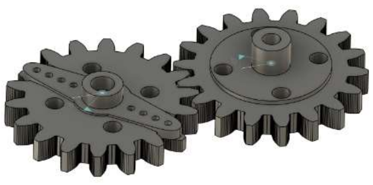
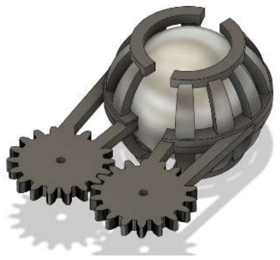
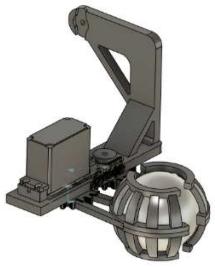
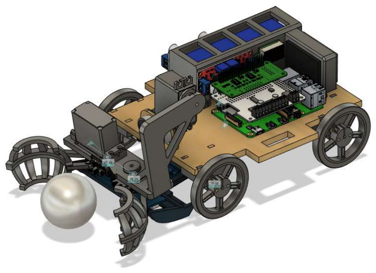
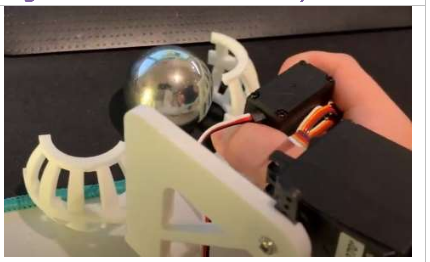
- Updated the full model (Fig. 12)

- Printed full set (Fig. 11), as well as a bracket to connect the servo and lifting system

- Tested claw mechanism – (VIDEO)

Comments

Initial gear design did not properly fit, holes are also unnecessary. Fixed the issue, then printed (Fig. 10)

 ***Figure 8** – Initial mock-up of claws*	 ***Figure 9** – Test gears*	 ***Figure 10** – Claw Assembly*
 ***Figure 11** – Claw assembly with bracket for lifting system*	 ***Figure 12** – CatBot Model w/ Claw*	 ***Figure 13** –Claw mechanism test*

24/4/2023 to 27/4/2023

Tasks Completed

- Created circuit diagram for all components (Fig. 14)
- Hooked up components in basic power setup (Fig. 15)
    - Used a DC motor to simulate load
    - Wiring works as expected, can continue and finish assembly
- Designed cage with gate at the back (Fig. 16)
    - Measured required height from ground, and used main model to adjust positions

Issues after print	**Resolution**
- Gate does not align with actual servo	Will redesign and fix model, then leave for now.
- Missing hole for servo	This part isn't necessary until attempting evac, so
- Holes need to be 3.8mm for heat inserts	best to leave it in case other changes are needed.
- Screws for display should be countersunk	If assembly is required, will leave servo off so it fits.

- Designed attachment for claw mount (Fig. 17)
- Finished design of first base (Fig. 18), and printed (Fig. 19) – Used model to figure out hole positions
    - **Issues:** Accidentally printed ultrasonic mount on the wrong side – holes won't be aligned
    - **Solution:** To reduce filament wastage, we cut it off with an oscillating multitool and then re-glued it
- Printed and prepared all parts for initial assembly.

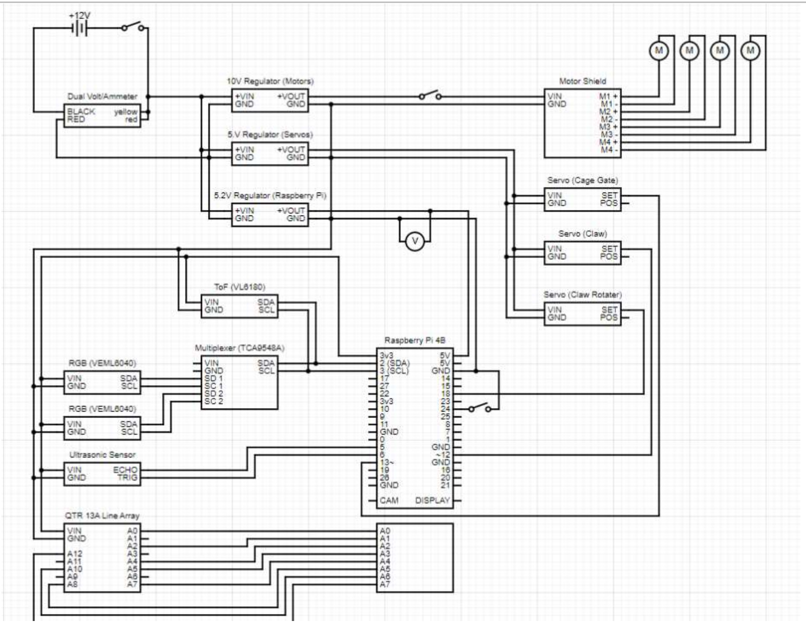


Figure 14 – Circuit Diagram

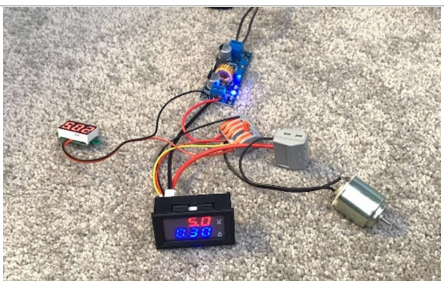


Figure 15 – Test power setup

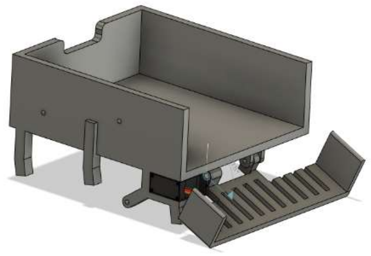


Figure 16 – First print of base

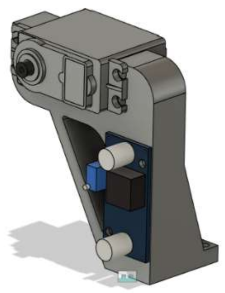


Figure 17 – Claw mount attachment

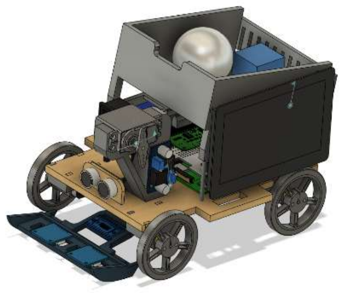


Figure 18 – Final first prototype

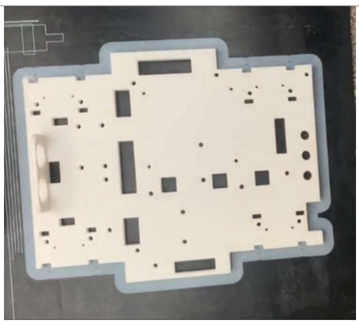
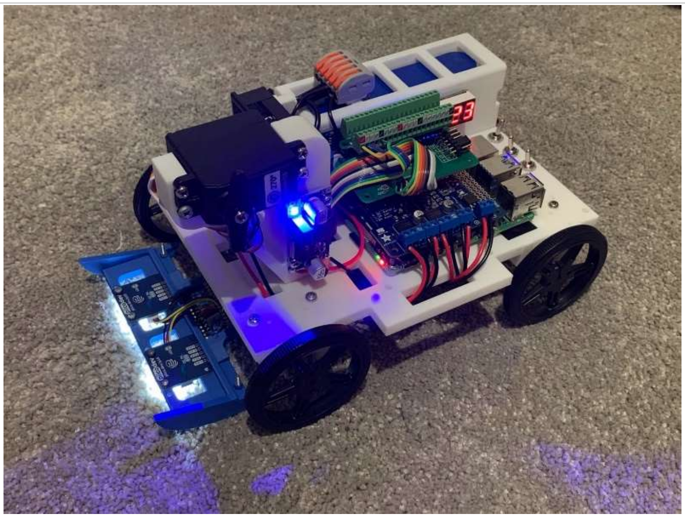


Figure 19 – First print of base

28/4/2023

Tasks Completed

- Fully assembled and wired robot prototype with all components (Fig. 20)

New issues found during assembly - Holes for motor brackets need adjusting ○ Inside two need to be 3.6mm ○ Holes under battery need to be countersunk to avoid hitting battery - Holes for colour sensors need to be slightly thicker, heat inserts warped the plastic. ○ Use thicker material for heat insert locations, especially for line array - Primary volt/ammeter needs to be moved forward slightly for better wire clearence - Lipo Holder holes for voltage regulators were way too small **Resolutions: These problems will be looked at when we need to next redesign the base.** *(Currently not critical enough to warrant a reprint)*	 ***Figure 20*** *– First full assembly, excluding cage and screen*

29/4/2023

Tasks Completed

- Set up Raspberry Pi
- Created GitHub repository: https://github.com/ZMcWilliam/catbot-rcji/
- Researched and found libraries and documentation relevant to all sensors and controllers
  - Motors: https://pypi.org/project/adafruit-circuitpython-motorkit/
  - Multiplexer: https://pypi.org/project/adafruit-circuitpython-tca9548a/
  - Colour Sensors: https://pypi.org/project/piicodev/
  - ADC Pi: https://github.com/abelectronicsuk/ABElectronics_Python_Libraries
  - Git: https://pypi.org/project/GitPython/

- Created motor helper library to provide simple wrapper functions around the motorkit library
- Created functionality test scripts for the following:
  - Motor control
  - PiicoDev VEML6040 Colour Sensors (VIDEO of test)
  - Line Sensor Array / ADC Pi

- Initial motor movement tests showed some issues when turning

Test	Result	Changes?
1. Moving straight (VIDEO)	Success	
2. Turning on the spot (VIDEO)	Not perfect, slides around a bit	Attempt thicker wheels
3. Turning w/ new wheels (VIDEO)	Lots of resistance, not any better	Accurate on the spot turning may not be entirely required.
4. Going up ramp, old wheels	Could not get up, resulted in slip	Add thicker front wheels
5. Ramp test 2 (VIDEO)	Success	Can struggle a bit on angles…

Conclusion: <u>Wheels will need some work</u>, but we will return to this later on as getting line following working first will help us choose what wheels to stick with. Returning to original thin wheels for now.

30/4/2023

Tasks Completed

- Implemented basic line follower program (VIDEO – Very minimal tuning)
  - Compiled all functionality test scripts into **follower.py**, and implemented a basic PID follower as well as a function, **calculate_position()**, to convert the values read from the line array into a usable input.
  - Most of the logic was cloned from the old CatBot, except converted into Python from C++
  - See code for relevant algorithm and comments – (COMMIT LINK)

Problem: The line follower cannot respond to the line as well as normal
Likely Cause: The sensor array is further forward from the steering axis than normal due to the 4WD
Solution: As seen in Figure 21, we moved the line array more towards the centre of the robot

Problem 2: Read rate from the line array is limited, additionally, we can only use 8/13 of the sensors
Likely Cause: Looking deeper into the sensor and library, it is limited to 0.00416 seconds per sample
- This rate, with 8 sensors, limits us to about 33 full sensor samples per second, in reality, it's about 10…
Solution: The ADC Pi's limitations have caused this, using 3 ADS1015's should solve the issues. Ordered.

- After moving the line array back as well as a little bit of tuning, the line follower was much smoother (VIDEO)
  - Having the sensor array moved back also means the robot is smaller overall, visible in Figure 22

New Problem: During the operation of the robot, we may accidentally forget to keep an eye on the battery level, this can be very problematic as if the LiPo gets too low, it will fail and be completely unusable.
- The voltmeter on the previous CatBot was the only way to monitor battery level, it is even harder to see on this new robot.

Solution: Create a battery alarm that creates a loud sound when battery gets low

- We created a mock-up of a battery alarm as seen in Figure 23 using some basic components from Jaycar.
  - https://www.jaycar.com.au/low-battery-alarm
- Another option was also found after some research (LINK) – This might be more reliable, so we ordered one

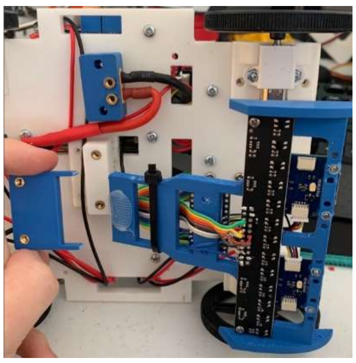


Figure 21 – Cut sensor array to move it backwards

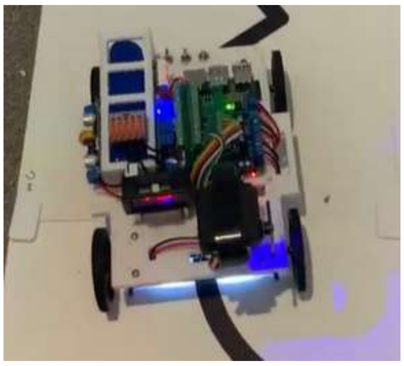


Figure 22 – Sensor array moved back during line follower

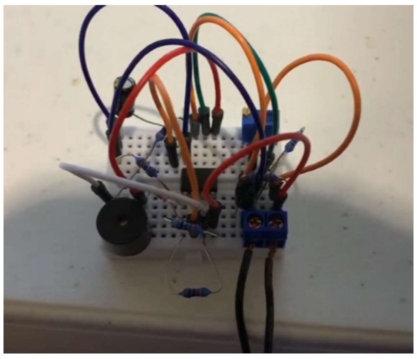


Figure 23 – Mock-up battery alarm

Week of 1/5/2023

Tasks Completed

- Started work on basic obstacle avoidance (VIDEO)
  - Added ultrasonic handling to the main program
  - Made some improvements to the motorkit helper

Issue	Resolution
Hard to keep track of the size of the obstacle, as it can be more than just a bottle now	Added a second ultrasonic sensor on the side of the robot
Reading from sensors is slow, and blocks up the main program	Added threading to the program Created a class for each sensor monitor thread
Motors would keep running after program stop	Added a check for interrupt to send motor stop cmd

  - By using the side ultrasonic sensor, and monitoring the distance, we can adjust the rate of steering to always stay a consistent distance away from the obstacle.

- Improved colour handling by adjusting thresholds for each sensor individually
- Added stop on red line check (VIDEO)
  - If both sensors see red, stop, wait a second, check again, if they are still red, end the program.

- Battery level alarm arrived, assembled on robot (Fig. 24)

Major Issue: When turning, wheels can get stuck on minimal edges and confuse robot
Cause: 4WD causes side-on collisions with the wheels
Solution: Try out different wheel options, and possibly change the steering axis to near the front again

- Attempted wider front wheels and Rotacasters for back wheels configuration, (Fig. 25) (VIDEO)
  - This works well and resolves the first issue
  - The steering axis is now between the front two wheels
  - Something weird happening trying to get up ramps, could be program related, however.
- Attempted mecanum wheels, which would also allow the robot to move sideways (Fig. 26) (VIDEO)
  - This causes several issues when trying to get up ramps, as robot can slide diagonally feely. (VIDEO)
  - This would likely work quite well for a maze bot, but unfortunately not rescue line.
- **Result**: Going to stick with the original option for now, and investigate again later, we want to focus on line following a bit more first as the result of that could significantly change what we need in terms of wheels.

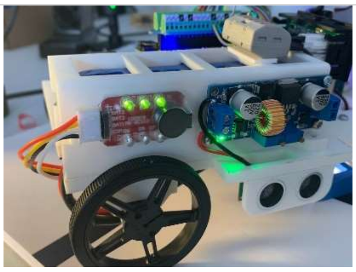


Figure 24 – Battery level alarm

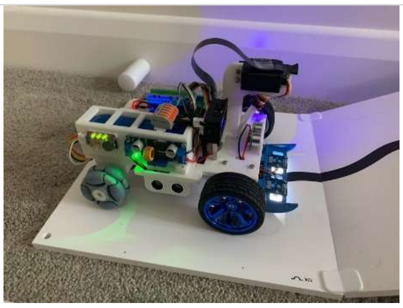


Figure 25 – Rotacaster wheel test

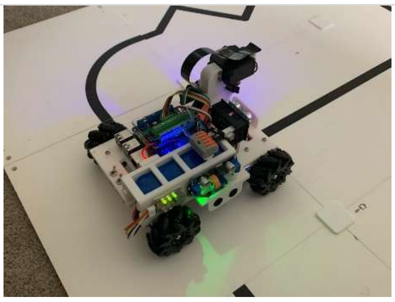


Figure 26 – Mecanum wheel test

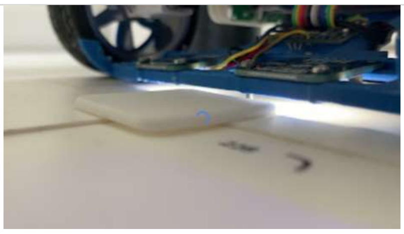
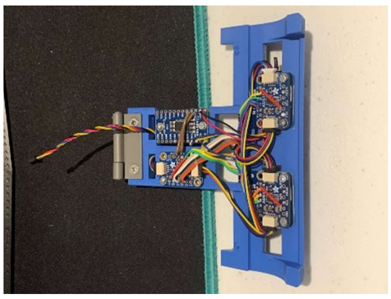
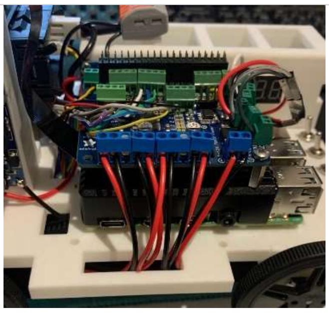
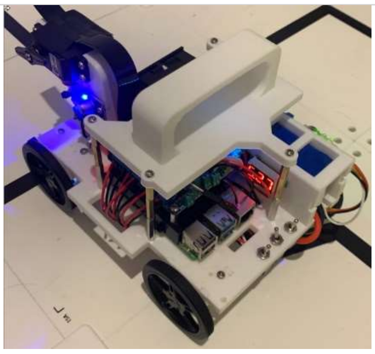
Tasks Completed

- Identified issue with sensor array bumping into objects in the middle (Fig. 27)
    - This could be quite problematic and cause the entire robot to get stuck
    - **Solution:** Redesigned sensor array to include slopes in as many places as feasible
    - **Additional Fix:** Sensor array needed to be redesigned anyway for new ADS1015's

- Printed and assembled new sensor array (Fig. 28)
    - Reduced output wires down to just 4 (Ground, 3v3, SDA, SCL)

- Added new heatsink shield for proper cooling (Fig. 29) (LINK)
    - Also removed need for terminal block hat by wiring it all on the motor shield itself
    - This required re-wiring the motors, as wires needed to be slightly longer

- Designed and added a temporary handle (Fig. 30)

 ***Figure 27*** *– Sensor array hits an object if it is small enough*	 ***Figure 28*** *– New assembled array*	 ***Figure 29*** *– New pi shield wiring and cooling hat*
 ***Figure 30*** *– Temporary Handle*		

13/5/2023 and 14/5/2023

- Designed and added a temporary handle (Fig. 30)

New Problem: Reading from the ADS1015's is a lot slower than expected
Solution: The i2c bus was limited in speed, we were able to get a 30FPS increase by increasing the baudrate
Notes on this are available here: https://github.com/ZMcWilliam/catbot-rcji/blob/master/NOTE-i2c.md

- Added ability to follow an inverted line (white line on black)
- Improved distance sensor monitoring by utilising the GPIOZero library
- Modified obstacle avoidance algorithm
  - Now turns and goes forward until the side ultrasonic sensor loses the obstacle
  - Repeats the above, with some other checks, to make its way around any sized obstacle
  - **Issue:** Requires 90 degree turns, wheels aren't always accurate enough
    Solution: Add a compass, after looking into a few options we decided on the CMPS14

- Lots of further testing and attempting to resolve problems with the line follower
  - Colour sensors are no longer reading consistent or accurate values
  - Line array is very good at missing data, or seeing something that will confuse the robot based on its position. Including completely losing track of the line if it is in the direct middle (see Fig. 31) or ending up in a bad spot on the line, and not knowing how to continue (see Fig. 32)

  - Due to this, and many other issues, we have decided to switch to a camera for line following. Zac has had some limited experience attempting this with a different bot last year, and we hope there will be enough time to fully code the bot using a camera to line follow.

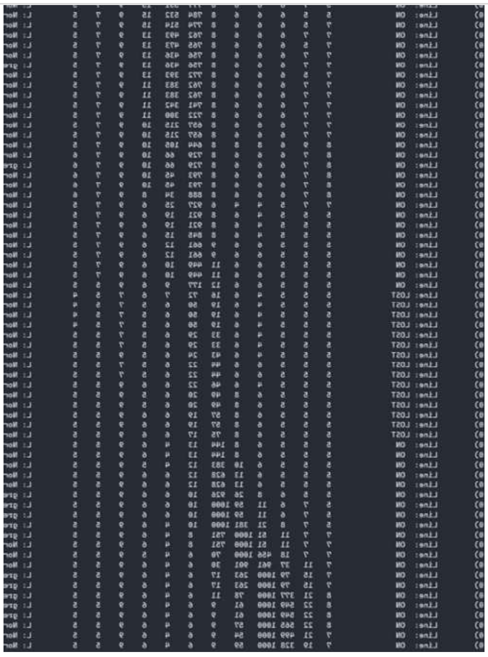


Figure 31 – *Program output when lost line in centre*

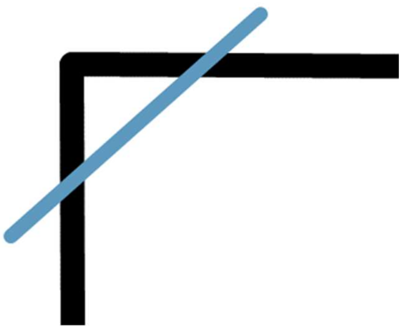


Figure 32 – *Example situation of the sensor array in a confusing situation if the robot turns the wrong way*

Week of 15/5/2023

Tasks Completed

- Created a mount for the Raspberry Pi camera and lens (Fig. 33)
- Redesigned servo holder to fit the new servo (Fig. 34)
- Printed off and mounted camera to the robot, removed line array (Fig. 35)

- Added camera test code using the picamera2 library

- Added CMPS14 compass and integrated into obstacle avoidance code
    - Connecting the CMPS14 required changing its i2c address as it conflicted with the motor shield An explanation of this is included on the repo (HERE)

- Started work on camera vision
    - To make this easier, we created a new temporary master file, **follow_camera.py**
    - Another file was created, **helper_camera.py** which sets up the initial camera and provides an accessible stream of images.
    - Working together with members of another team from our school, Yuma and Seb were able to provide some helpful information as to where we can get started.
    - Some simple functions were created to isolate the image (Fig. 36) and convert it to grayscale, find a binary to use for the line, and create a mask for the green visible in the image. (Fig. 37)

- Added white LEDs (Fig. 38) to the robot to reduce shadows and provide a better image
    - As seen in Fig. 36, a significant shadow is visible, in Fig. 37, this is no longer an issue.

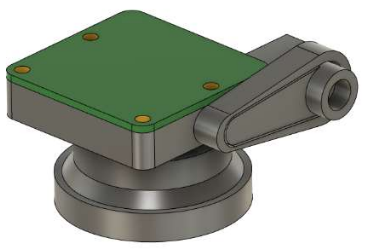


Figure 33 *– Pi camera mount*

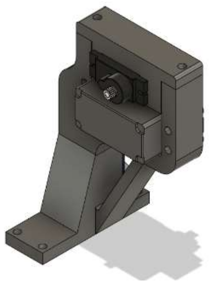


Figure 34 *– New servo holder*

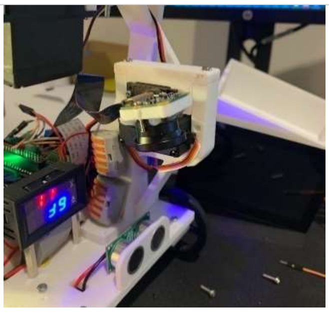


Figure 35 *– Cam mounted on bot*

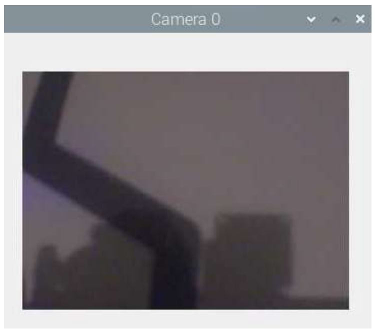


Figure 36 *– Camera View*

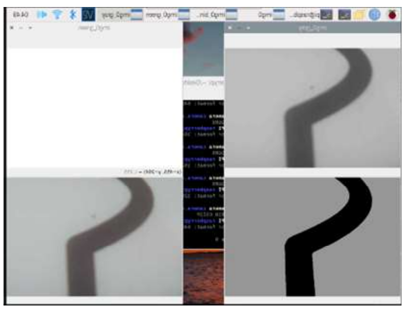


Figure 37 – *Other converted camera views*

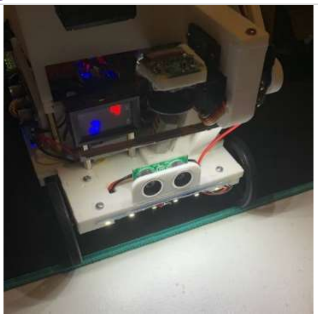


Figure 38 – *Added LEDs*

23/5/2023

Tasks Completed

- Fixed various issues with the camera image quality

Issue: Blue on the camera was showing as yellow (Fig. 39)
Cause/Fix: The values from the camera were in the order BGR, converting this to RGB resolved the problem.

Issue: Thresholding is difficult due to varying light levels (Fig. 40 left)
Fix: Added a calibration script for white, then used this to scale all image values (Fig. 40 right)

- Started to try and threshold green out of a HSV converted image
  - **Issue:** It is difficult to calibrate the max and min values required for thresholding
  - **Solution:** We created a UI using Tkinter to easily adjust these values as required (Fig. 42)
  - As can be seen in Figure 42, this new UI has allowed for fine-tuning and resulted in a clean output

- Redesigned cage, and added proper handle with CATBOT NEO text (Visible in Fig. 43)
  - As concluded after the first print, the cage was not viable and did not fit the servo. We redesigned this to fix these issues so that all servos could be assembled.
  - Wired up every servo on the bot
  - Added a handle to the cage itself for easy carrying

Issue: As seen in Figure 43 and 44, when the bot is looking down a ramp, it struggles to see properly
Cause: The light level down the ramp drops significantly
Proposed Fix: Ignore these values, it could possibly be treated as a really deep intersection? Will attempt later

- Added tests for servos (VIDEO of all servos moving)
  - It is critical for the camera to be at a consistent position, so we set up servo controlling code
  - The servos were initially quite jittery, but by using the GPIOZero library along with the default pin factory set to use PiGPIO. This is documented in the readme

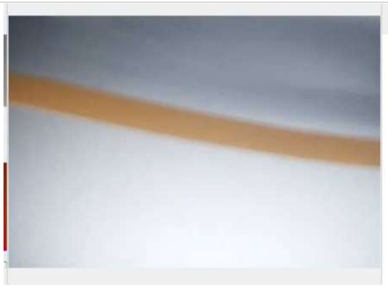


Figure 39 – Blue looks like yellow

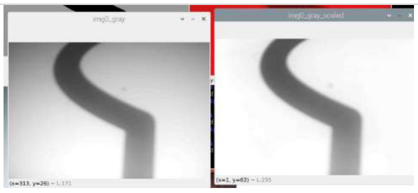


Figure 40 – Scaled image before and after white calibration

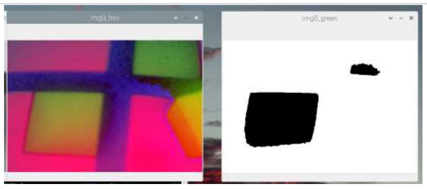


Figure 41 – HSV Thresholding

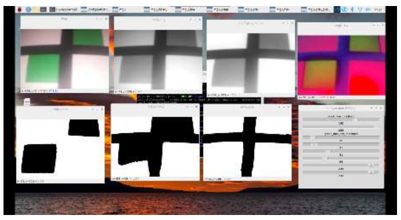


Figure 42 – GUI (Bottom right)

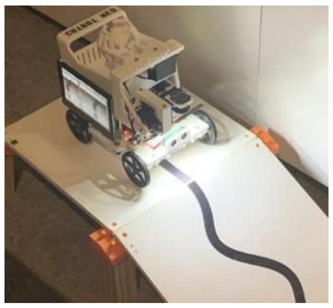


Figure 43 – Bot looking down ramp

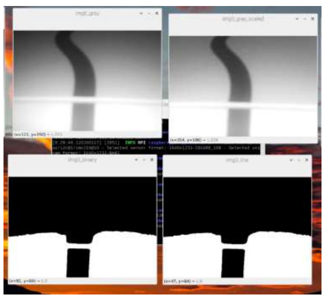


Figure 44 – Perceived values on top of ramp

24/5/2023 and 25/5/2023

Tasks Completed

- Started work on algorithm for processing the line
  - The first step is to find the major white and black contours, see Figure 45
  - From here, we can use an algorithm to select the most preferred black contour (assuming multiple)
    - This filters out contours that are too small, and sorts the remaining contours based on their distance from the last known line. This prevents accidentally flipping between contours.
  - Then, we retrieve the bounding box of the chosen contour, and figure out its angle and error (distance from the centre of the image) – Figure 46
  - To prevent the angle from being 90 degrees off, we can use some additional logic to figure out when to flip or offset the angle value.
  - After the desired angle and error have been calculated, this can be used as an input to a new PID function, which will require very different tuning than before – yet to come.

  - Some more fine tuning is required, but this has resulted in a decent initial follower.

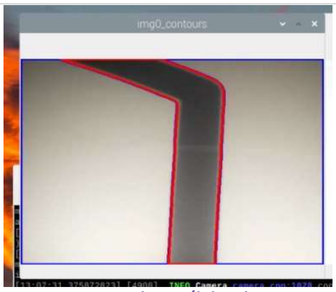


***Figure 45** – White/black contours*

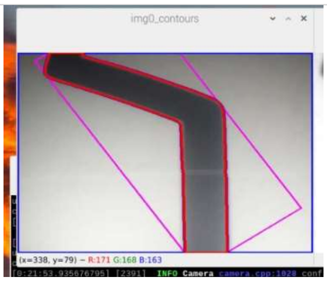


***Figure 46** – Scaled image before and after white calibration*

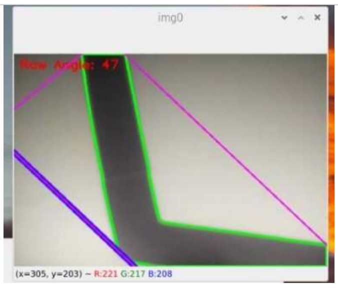


***Figure 47** – Example processed line*

28/5/2023

Tasks Completed

Issue: The camera struggles to deal with intense steering
Cause: The 4WD sets the camera very far forward compared to the steering axis
Fix: Use the Rotacaster wheel method established before to move the axis to the front but keep 4WD

- Replaced back wheels with Rotacasters (Fig. 48)
- Reprinted base (Fig. 49 assembled)
  - Moved everything at the front as far back as possible, to allow camera to see further under robot
  - Adjusted wheels to be closer to the centre of the robot to help prevent falling off a tile
  - Fixed some other issues based on what we found after the first iteration
- Steering axis is now as close to the camera as possible.

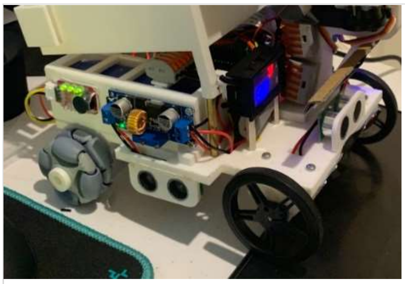


***Figure 48** – Added Rotacasters*

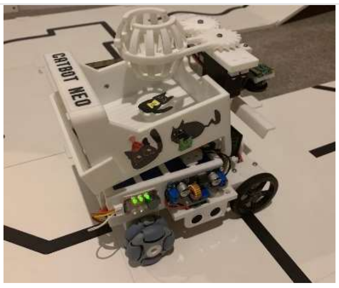


***Figure 49** – Assembled base V2*

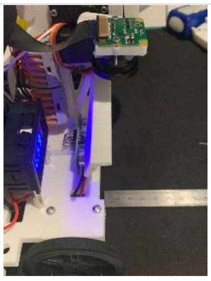


***Figure 50** – Camera view*

29/5/2023

Tasks Completed

- Attempted to add black calibration (unsuccessful)
    - The LEDs cause black to be inconsistent, we had an idea to calibrate black as well as white
    - Calibrating both led to a rather strange result, it's not really possible to use both as maps at the same time easily. See Fig. 51 for an example of what the camera might see after this calibration
    - We decided to stick with just white calibration, this isn't that big of an issue.

- Started work on handling intersections
    - There are several parts to handling intersections, we can classify them based on how many white contours we can see, for example:
        - **4 White Contours:** Robot is likely in the middle of a 4-way intersection (Fig. 52)
        - **3 White Contours:** Robot is likely in the middle of a 3-way intersection (Fig. 53)
        - **2 White Contours:** Robot is at no intersection, OR is entering/exiting an intersection (Fig. 54)

    - To start, we decided to attempt handling entering and exiting an intersection (2WC)
        - Each contour can be simplified into a set of points using **cv2.approxPolyDP()** (Fig. 55 blue)
        - By processing each of these points, we can find the edges of the line we want to follow, and cut everything else out (Fig. 55 yellow)
        - **Issue:** This method does not work for curved lines
          Conclusion: We will come back to 2WC later and focus on 4WC first, as that should be easier

Issue: Green on the image was affecting intersection code
Cause: The intersection code was using the wrong binary image as an input
Fix: Switched to use img0_line instead of img0_binary for intersections (Fig. 56)

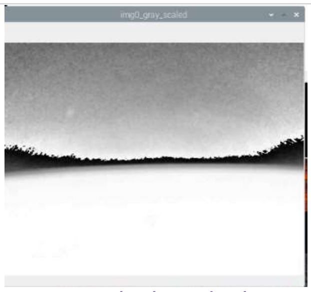


Figure 51 – *Black and white calibration test*

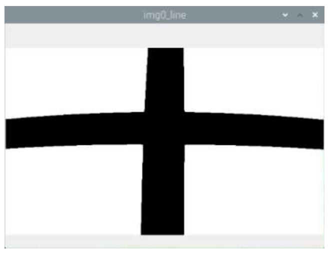


Figure 52 *– Example intersection with 4 white contours*

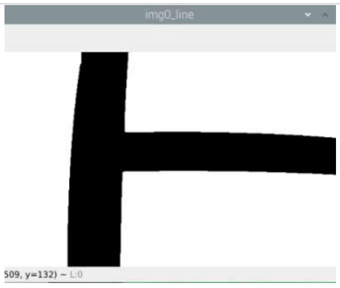


Figure 53 *– Example intersection with 3 white contours*

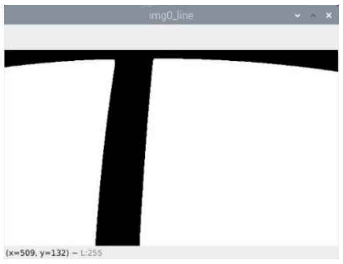


Figure 54 – *Example intersection with 2 white contours*

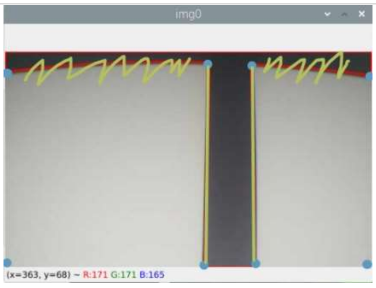


Figure 55 *– Representation of 2WC simplification and cutting out data*

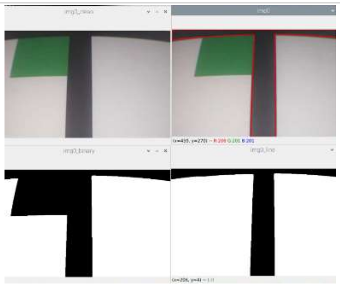


Figure 56 *– img0_line is used for intersections to ignore green*

30/5/2023

Tasks Completed

- Added handling for 4 white contour intersections
  - Similar to 2WC, the goal is to find points to use as a mask for removing unwanted parts of the line.
  - The major points we want to find are the 4 closest to the centre of the intersection (Fig. 57)

  - To find these points, we first simplify each white contour down to 4 points like before.
  - Then, we find use the midpoint of each contour to find the middle of the intersection (Fig. 58)
  - Knowing the middle, we find the closest point in each white contour, resulting in 4 optimal points.

  - These 4 points let us make a mask to get a new image with only the line we want to follow (Fig 59)

  - As always, more documentation has been included in the source code

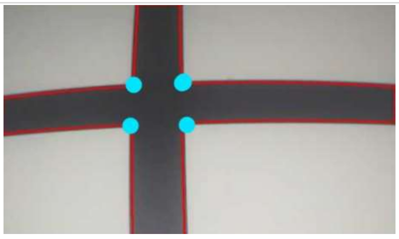


***Figure 57 –** Optimal 4WC points*

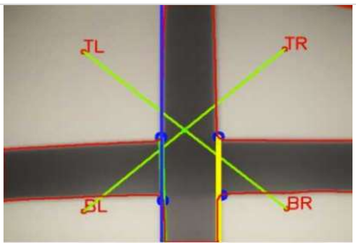
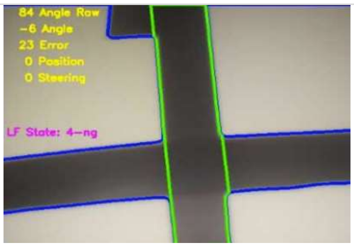


***Figure 58 –** Example intersection with 4 white contours*


```
84 Angle Row
-6 Angle
23 Error
 0 Position
 0 Steering

LF State: 4-ng
```
***Figure 59 –** Processed intersection*

31/5/2023

- Obtained hard case with plucked foam for trip (Fig. 60 and 61)

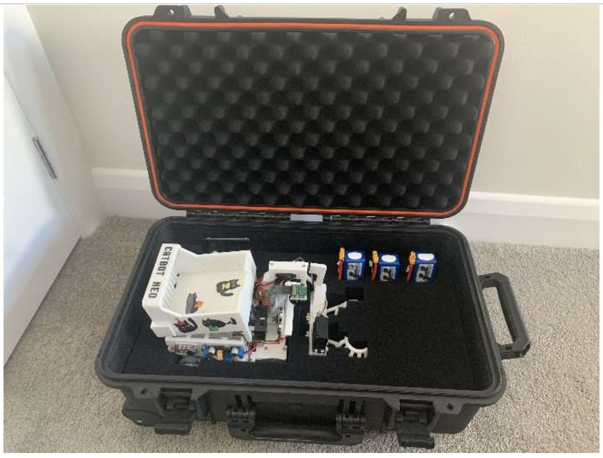


***Figure 60 –** Case with foam layer 1*

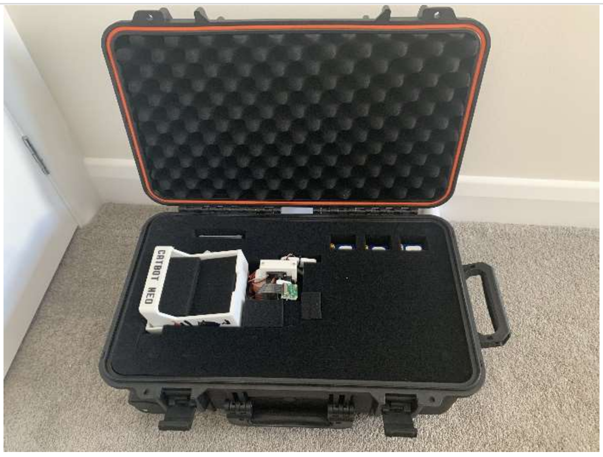


***Figure 61 –** Case with both layers of foam*

June

1/6/2023 to 3/6/2023

Tasks Completed

- Fixed **centreOfContour()** function (evident in Fig. 63)
  Issue: The centreOfContour function was not finding the centre of a contour properly
  Cause/Fix: Was dividing by 3 instead of 2… Completely rewrote to use **cv2.moments()** instead

- Created a setup to make testing intersections easier
  - We found that testing intersections required lots of rotating of the robot, using an iPad linked to a laptop running a simple website Zac made let us to have CatBot just sitting on the table (Fig. 62)

- Added handling for 3 white contour intersections
  - Three white contours require a bit more complicated logic compared to 4WC.
  - A similar algorithm is used to find two points (yellow in Fig. 63) and use it as a cutting mask (Fig. 64)

  - **(Issue)** We now must determine which direction to apply the mask in, otherwise you get Fig. 65
    - The first attempt at fixing this was to determine which sides of the screen each contour touches and use some logic to match that in a direction. This was very complicated
      - You can see the start of us figuring this out in Figure 66.
  - **(Fix)** Luna came up with a simpler solution, we can determine the cut direction based on the side of the line that has the most contour centre points.
    - By implementing this, along with some exception cases, we achieved a good result! (Fig. 67)

- Fixed some further issues with 3 white contour intersections (COMMIT)
  - **Issue:** When entering/exiting, points found could touch the top, leading to inaccurate cutting masks.
    Fix: We now check for this and find the next best point to use instead

  - **Issue:** The cut direction could end up the wrong way if the contour area is very small
    Fix: Added some checks for small contours to swap the direction if needed

  - **Issue:** CutMaskWithLine() failed if the line was flat
    Fix: The easiest fix was the move one of the points upwards by 1 pixel

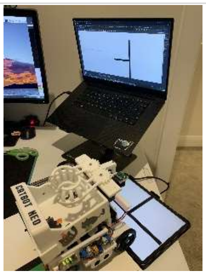


Figure 62 – iPad setup for easier testing

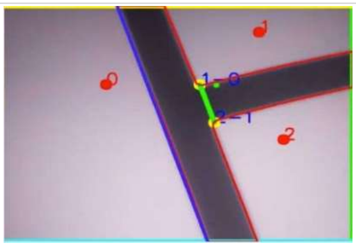


Figure 63 – Processed 3WC points

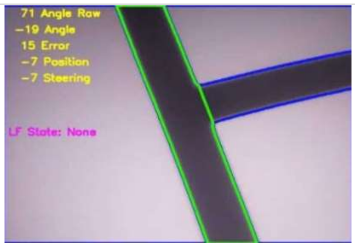


Figure 64 – Processed intersection

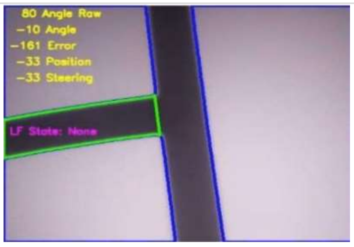


Figure 65 – Problematic intersection processing

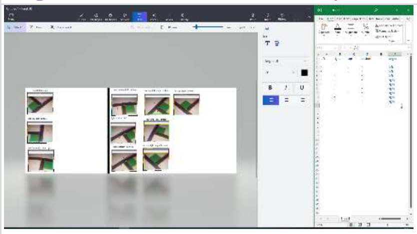


Figure 66 – Starting to figure out fix for 3WC cut direction

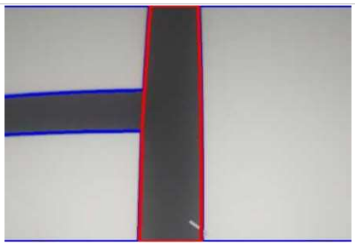


Figure 67 – Fixed intersection

4/6/2023 to 5/6/2023

Tasks Completed

- Improved entering and exiting intersections (2WC)
    - Various improvements were made to this code, mainly focusing on what to do in each edge case of intersections.
    - It has been established that this code is rather slow and resource intensive to run every loop, we will return to this later and try to optimise it.

- Removed intersection debugging code as it is no longer very useful.

- Moved TKinter calibration code into a separate, **calibrate_cam.py** file to reduce code size in the main file.

- Added FPS tracking info to start looking into optimisations.
    - CatBot is running at about 16 FPS, it should be possible to achieve much higher

Issue: The camera view was not centred **Cause/Fix:** Unnecessary cropping was being applied horizontally, removing this fixed the issue

Issue: When no contours are found, the bot will lock itself to going straight **Fix:** Added tracking for the current steering value, it will now retain the last known value upon losing the line

10/6/2023 to 13/6/2023

Tasks Completed

- Added FPS balancer.
    - When the main program FPS is higher than the camera FPS, we are not getting any useful frames out of it. To avoid this, we now have a balancing system which will add delay to the main program to try and give more processing power to the camera system.
- Moved main image processing to camera thread.
    - All frames we read require certain processing into resized, grey, binary, green, etc versions.
    - Moving this to the camera thread means the main thread can be freed up a bit more.
- A large cause of low FPS is the 2WC code, we have disabled it and will need to make a new version.

Issue: cv2.imshow() adds a fixed delay, reducing available FPS if we want to use it **Proposed Fix:** Move image previews into a thread. **Result:** imshow is not thread-safe, this doesn't work. We will continue to have to disable all img views

- Added circle detection for Evacuation Zone
    - By utilising cv2's **HoughCircles** function, we were able to start working on the Evac Zone (Fig. 66)
- Added a MOTD when connecting to SSH (Fig. 67)

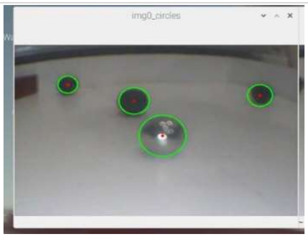
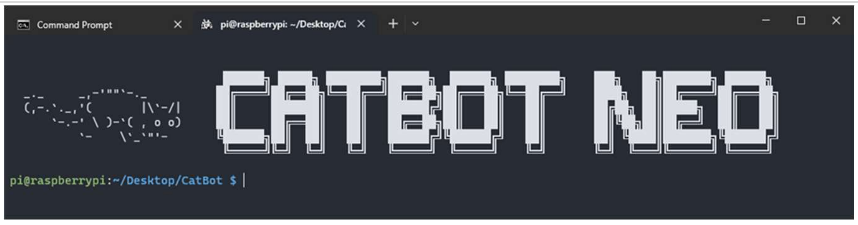


Figure 66 – Circles preview



```
CATBOT NEO

pi@raspberrypi:~/Desktop/CatBot $ |
```

Figure 67 – New SSH Splash text

14/6/2023

Tasks Completed

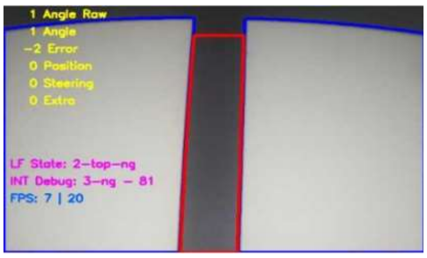
- Fixed problems with image processing
  - Images were not scaling correctly to calibrated white.
  - Processed line image was not removing green.
  - Black line was appearing at the bottom of the image (Fig. 68)

  - **Cause:** Uncertain, this code worked in the past, not worth spending time investigating

- Recreated and optimised 2WC algorithm (Fig. 69, Fig. 70)
  - Now instead of constantly cropping out the image, we check if there is significant black near the top or bottom, assisted by knowing what previous intersection state we were in.
  - If all points in the white contours are not touching the top, but are near the top, we could be entering an intersection.
  - We can then use this information to completely cut out the top or bottom up to 80px when required

Issue: "Clean" images were showing evidence of modification, such as text
Cause/Fix: Python links back modifications to arrays, using .copy() on the array resolves this problem

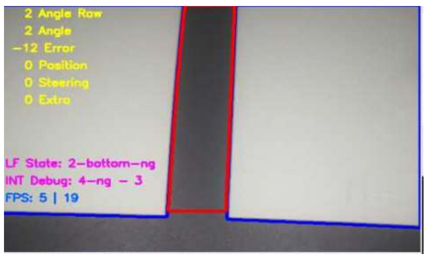
Figure 68 – Random black line on cropped image preview



```
1 Angle Raw
1 Angle
-2 Error
0 Position
0 Steering
0 Extra

LF State: 2-top-ng
INT Debug: 3-ng - 81
FPS: 7 | 20
```

Figure 69 – Entering intersection



```
2 Angle Raw
2 Angle
-12 Error
0 Position
0 Steering
0 Extra

LF State: 2-bottom-ng
INT Debug: 4-ng - 3
FPS: 5 | 19
```

Figure 70 – Leaving intersection

15/6/2023 to 19/6/2023

Tasks Completed

- Focused camera and recalibrated accordingly. (Fig. 71)
- Added a debug switch to the line follower to disable/enable CV2 previews.
- Started work on green turn handling (Fig. 72)
  - Look for large enough green contours, and then identify which ones touch the bottom of the screen
  - Currently just draw the identified contour – but will use this to draw a new line to follow soon
  - If there are 2 matching green contours, we know to turn around 180 degrees
- Finalised all documents (Journal, TDP, and Poster) for submission due on 19th of June

Figure 71 – Image after focusing – significantly less fuzzy

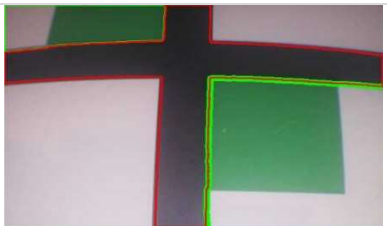


Figure 72 – Initial green turn handling mockup

20/6/2023

Tasks Completed

- Finished handling for single green markers
    - Using the centre point of each green contour, it is possible to identify which white contour the green relates to. This can then be used to figure out whether the green turn is in below the intersection and should be followed. Fig. 73 shows an initial attempt to cut out a new line by masking only what is immediately surrounding the selected white contour, however, this didn't work particularly well when approaching from non-straight angles.
    - Instead, we found that the most effective way to create a new followable line was to effectively dilate the selected white contour by drawing a new thick line around it, and use that as a mask over the black/white line image, this is the green/red line found in fig 74 and 75
    - While the robot turns, it is possible that multiple white contours touching the bottom of the screen will also contain a green marker (Fig. 75). To prevent accidentally switching to the wrong one, we track what direction we are currently turning after starting a turn, and favour the leftmost or rightmost contour, which works quite reliably.

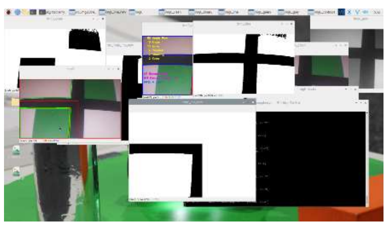
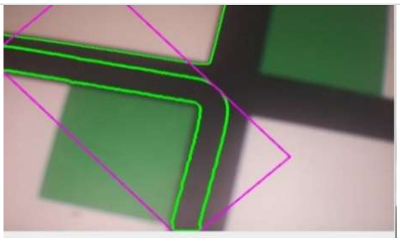
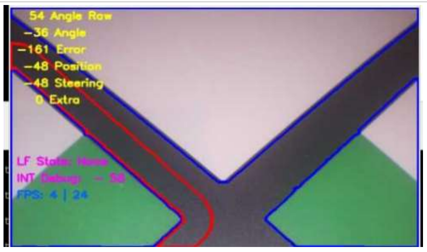
		
		```
		54 Angle Raw
		-36 Angle
		-161 Error
		-48 Position
		-48 Steering
		0 Extra
		LF State: None
		INT Debug - 58
		FPS: 4 | 24
		```

Figure 73 – *Proof of concept line to follow based on green*	***Figure 74*** *– Final new followable line contour (red) for green turns*	***Figure 75 –*** *Distinguished which green contour to continue following*

21/6/2023 to 24/6/2023

Tasks Completed

- Improved line following with significant turns – Steering was not effective on right angle turns as the angle was determined by the angle of the line's bounding rect, in this case, a square with near 0 rotation (Fig. 76)
    - The calculation for this now uses the bottom left and top right points with a fixed offset
    - If the angle of the contour is large enough, we also turn a bit more to prevent loosing the line

Issue: The image stream from the camera keeps permanently freezing at a random point in time after the program is started, from anywhere between a minute to 10 minutes or more **Cause:** We spent a few days looking into why this is would be occurring, after attempting to replace the camera, cable, and Raspberry Pi, we had no luck, so instead a workaround was developed **Fix:** The code to capture each image from the buffer is now in a separate thread and actively monitored from the main CameraStream loop, if no image has been added in the past second, the entire thread gets recreated. This seems to work for now, but we will have to look at different solutions in the future

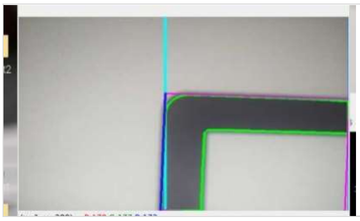


Figure 76 – *Bad angle to follow (cyan) on a 90 deg turn, calculated based on the rotation of the purple bounding box.*
This angle should optimally be calculated from the bottom left to top right corners of the bounding box.

25/6/2023

Tasks Completed

- Reorganised and improved code structure
    - Moved functions such as centerOfContour or simplifiedContourPoints to a separate file
    - Moved global variables around in sections that make more sense
    - Added various comments

- Merged features from the legacy follower script (pre-camera) to integrate with the camera code
    - This includes code for devices such as the ultrasonic sensors and compass, allowing us to work on these components again alongside the vision system
    - Also adjusted some debugger things to be clearer as to where they originate from

27/6/2023 and 28/6/2023

Tasks Completed

- Added stop on final red line functionality (Fig. 77)
    - Only checks for red every 5 frames to save on processing, as this should only appear right at the end of the course.
    - New config values have been added to threshold red from HSV
    - It has been identified that our current solution for config is beginning to get fairly cluttered (Fig. 88), we will likely start work on a more condensed and tabbed view for this shortly.

- Added a test for the VL6180 Time of Flight sensor, as well as integrating it into follower.py
    - Will be used to distinguish black/silver once a ball has been grabbed, and confirm successful capture

- Improvements to circle detection, and created a **rescue_zone.py** file to start testing the rescue algorithm without interfering with line following yet
    - Added a height buffer and checks to restrict size of balls depending on the height on the image they are located at, allows us to exclude random data that isn't a ball
    - This could be further improved by adding more restrictive heights in smaller intervals than just 2

- Added a graceful shutdown to the code so that all CV2 windows, the camera, etc, are properly destroyed

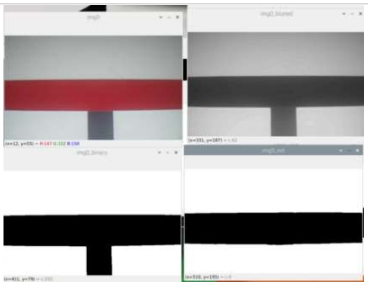
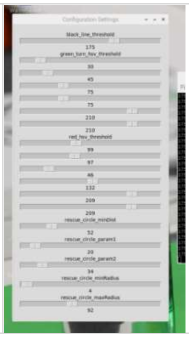


Figure 77 – Red Line Processing


Configuration Settings

black_line_threshold
175
green_turn_hsv_threshold
30
45
75
75
210
210
red_hsv_threshold
99
97
48
132
209
209
rescue_circle_minDist
52
rescue_circle_param1
20
rescue_circle_param2
34
rescue_circle_minRadius
4
rescue_circle_maxRadius
92

Figure 78 – Config interface is starting to grow significantly out of control in size

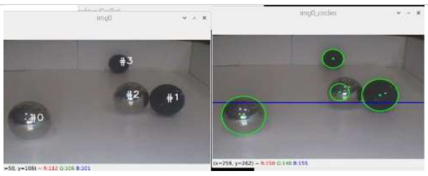


Figure 79 – Better circle detection with height buffer (blue line) to change check for size based on height

29/6/2023 and 30/6/2023

Tasks Completed

- Fixed motors not all running at the same speed when wanting to go "straight"
  - Each motor seems to have a slightly different speed compared to others, mainly the front left motor
  - Offsets have been added to ensure that a requested speed of "30" for all motors is consistent

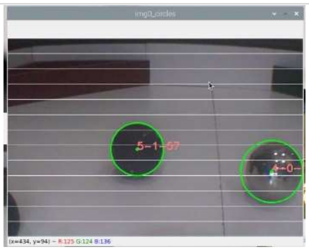
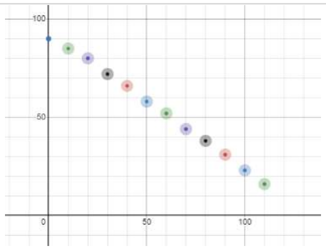
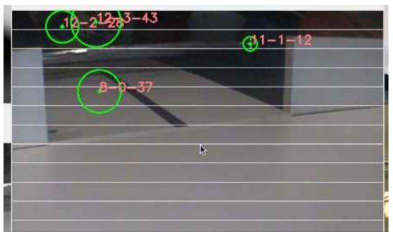
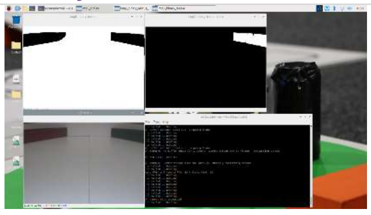
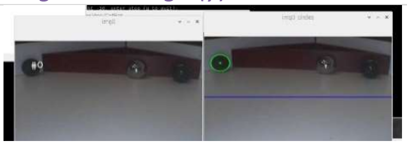
- Created a simple proof of concept script to make the robot approach any ball on the screen
  - Needs to be able to track if the circle has been lost instead of continuously approaching something that could have disappeared
  - Needs to figure out when the ball is close enough to do a final approach a bit better
  - There are still issues with mis detecting balls, this is primarily due to the nature of attempting to use Hough Circles to solve this problem, but we will do our best to work around it due to a lack of time…

- Added more height restrictions as to what radius a ball must be at various parts of the screen to be counted as a valid ball. See Fig. 80 and Fig. 81, this could be done with a linear function but for now we are just iterating over a list of preset values

- A major issue we are encountering is that the robot can easily end up seeing random stuff outside the evacuation zone, especially if it were to look at the entry or exit and see lines (Fig. 82)
  - To help avoid this issue, we have added some additional logic to look for at least 30 pixels of continuous white pixels from the top, anything above this is ignored.
  - This fixes the issue for the most part, but still struggles to cope with looking outside at a line…

- Created an algorithm to locate and approach the green evacuation block, currently ignoring the red as it is proving difficult to distinguish and is not important unless we solve the rest of evac first.
  - This is rather simple at the moment and just looks for a significant contour to target and approach

- A couple other modifications were made including rotating using a bearing to ensure we aren't stuck on something at the start, as well as tracking the qty of balls found before doing the block approach. Evac essentially works now about 3/5 times in a row, other improvements will be made

 *Figure 80 – Added more height bars to further restrict balls*	 *Figure 81 – Radius on ball (x) vs height on image (y)*	 *Figure 82 – Robot identifies a bunch of non-existent victims*
 *Figure 83 – Distinguished green evacuation block*	 *Figure 84 – Struggles to distinguish balls in front of red block*	

July

1/7/2023 to 3/7/2023

Travelling to London from Melbourne from 1st to 2nd of July – Some work was completed while on the planes
Note: *Luna was unfortunately unable to attend the competition in person but still collaborated where possible over the internet*

Tasks Completed *(Melbourne to Bangkok)*

- Improved graceful exit script to ensure it only runs once, it was getting stuck and not realising it already ran

- Fixed double green detection, if we started “turning” before realising there were 2 valid greens, it would fail to identify that a double green existed and treated it as if we wanted to continue turning.

Tasks Completed *(Bangkok to Muscat to London)*

- Created a program for obstacles that should be able to reconnect back to the line, unable to easily test
- Added logic to turn 180 on double green using compass
- Moved first frame wait prior to starting the loop to avoid checking it again
- Added a global debugger toggle inside program to turn on/off some debugging code
- Improved readme in git

Tasks Completed *(London)*

- Auto start code on boot
- Fixed issues with double green logic coded on the plane
- Camera servo seems to have subtly changed during travels, adjusted initial angle to accommodate

Figure 85 – *Working on CatBot in Melbourne Airport*	***Figure 86 –*** *CatBot on the plane between Bangkok and Muscat*	

4/7/2023 and 5/7/2023

Tasks Completed *(Paris)*
- Decreased default timeout for the bearing alignment to 10s instead of 1000
- Simplified obstacle avoidance
  - The side ultrasonic sensor was experiencing issues, so the robot no longer attempts to figure out the size of the obstacle, instead, it will just go around it at a fixed distance. This will likely decrease the reliability of the obstacle avoidance during the event, but it should be fine for the most part.
- Integration of new calibration script

6/7/2023

Tasks Completed *(Bordeaux, Pre-Event)*

- Calibrated all colours/settings for venue conditions (Fig. 87)
- Implemented evacuation zone program into follower program, all code is now together
- Improved detection of red stop, now confirms it actually exists and reverses a bit to prevent overshooting
- Added a janky solution to detect the evacuation zone
  - Looks for a T junction where approaching it leads to a dead end. The check for red is run first to avoid issues with detecting that as an evacuation zone entry
  - This still has issues, as the bot is barely able to cleanly mask out the silver foil (Fig. 88)
- Fixed some issues with the configuration data relating to evac
- Prevented image preview for no black contours from showing when not in debug mode
- Catch all errors in the main loop with a try except, to save the program just in case of some random issue

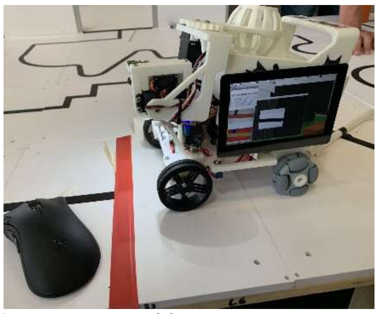


Figure 87 – *Calibrating at venue*

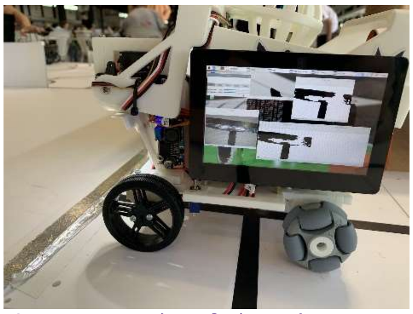


Figure 88 – *Silver foil readings*

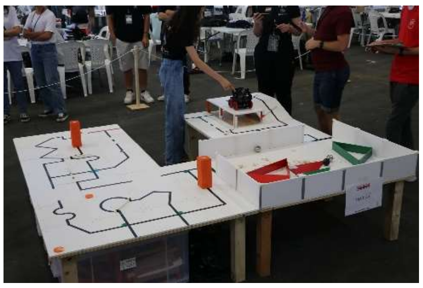


Figure 89 – *One of the courses*

Event Day 1

Tasks Completed

- Detect evac using obstacle and colour check (foil was too hard to detect)
  - The silver foil was proving very difficult for the camera to distinguish as it is very reflective
  - The robot would always follow the line to the left and then hits the wall before detecting an obstacle, this can be used to our advantage by then using the camera to figure out if an obstacle exists by checking for the existence of orange, if not, run the evac sequence
- Randomise obstacle direction (Fig. 89)
  - The obstacles used ended up being quite close to walls or other lines, given little time, I chose to implement a random direction approach so that hopefully it will eventually attempt it correctly
- Fix evac circle and block detection
  - Calibration was required due to the new venue. Some other issues were also identified and resolved
  - I accidentally let the robot use an old broken calibration for a few rounds without noticing… whoops
- Use ultrasonic to help with final block approach in the evac zone
- Fix double green causing the robot to fall off the tile
- Slightly fix gaps in the line
- Add increased speeds on ramps

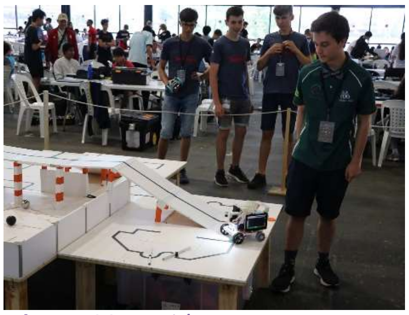


Figure 90 – *Mid Course*

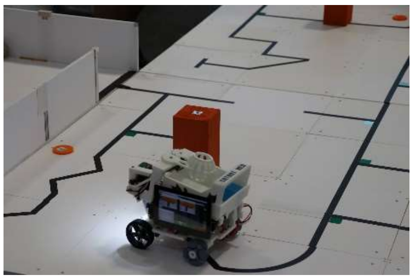


Figure 91 – *Obstacle*

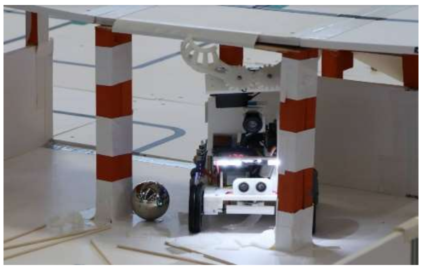


Figure 92 – *Stuck on a pillar*

Tasks Completed

- Fixes for evacuation zone detection
- Check for captured ball
- Skip to block finding if evac takes too long
- Rotate 270 degrees instead of 360 in evac entry
    - Going forward was a bit problematic in most situations, ending up 90 degrees rotated allows the robot to identify the wall and then get away from it
- Resize evac debug images
- Speed up evac approach
- More fixes for gaps in the line
    - Since CatBot is running rather slow in framerate for some reason, it is harder for the robot to distinguish a gap in the line from a really tight turn, both of these seem to be appearing on the courses, so it is hard to account for both
    - The bot now randomly chooses whether to hard steer or to go straight, similar to the obstacle solution. It's not a good solution, and if time permitted, this would need to be investigated more
- Run full speed after 18 seconds on a ramp
- Speed bumps were causing a lot of issues, the robot will now quickly jump forward if the bearing is consistently maintained for 10s to (hopefully) make it over
- The evac obstacle check was
- **Started work with team Apollo 12 (Israel) for the SuperTeam challenge**

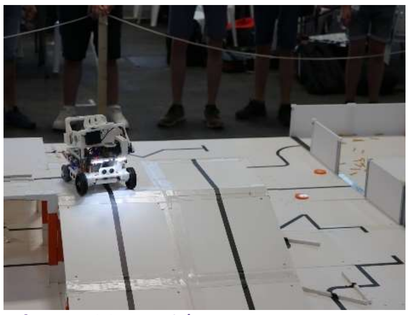


Figure 93 – Mid Course

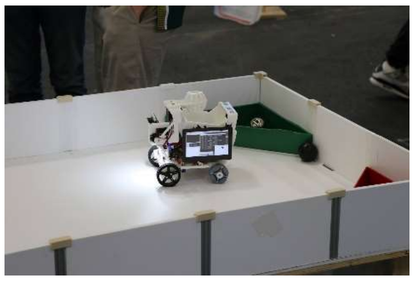


Figure 94 – Successful Rescue

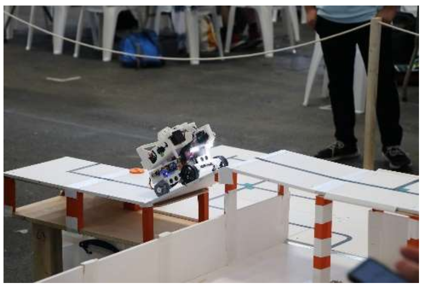


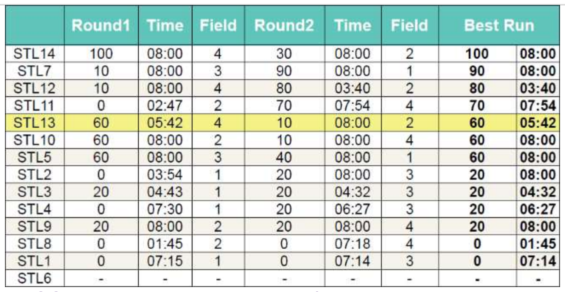
Figure 95 – Evac zone with debris

Tasks Completed *(Super Team Challenge)*
The aim for the SuperTeam challenge was to have two robots, one “Ambulance”, and another following behind, to each rescue 1 victim from the evacuation area. The ambulance robot must navigate through the course and use each intersection marker to know the “safe” route, at the same time, this robot should be communicating to the other so they can follow closely behind.

We chose to have CatBot lead as the “ambulance” robot, as it was more reliable at following the course.
Proposed Solution:
- Connect robots via Bluetooth (Apollo's HC-05 module to Raspberry Pi).
- CatBot gets a head start, and starts finding intersections
- After 10s, Apollo starts and finds the first intersection, then sends a request to cat for the first instruction
- CatBot sends back the side apollo started on, and which way to go at the intersection (forward or turn)
- CatBot will be far ahead of Apollo, so each time Apollo reaches an intersection, it will request for a direction
- Include checks that Bluetooth instructions were received and correctly read (add checksum bit)
- CatBot rescues silver ball first, then, since Apollo can't lift, pick up black ball and then place it on Apollo for it to be rescued.

What we were able to achieve:
- Due to extremely little time and BT issues, we were only able to achieve CatBot's following and rescue side.
- A modified program was created and is capable of solving the full line with every intersection, then rescuing a victim and exiting
- CatBot does not distinguish between black and silver victims yet (but is capable), we found that it would reject black for some reason anyway
- Apollo waited at the first intersection to avoid a lack of progress, unfortunately, this means we could only get points for the rescue
- In the first round, we successfully completed the entire line and successfully rescued one victim.

If we had more time:
- Apollo was capable of listening and responding to instructions, if Bluetooth had of worked.
- In this case, both robots should have been able to make it to the end successfully
- Since Apollo couldn’t lift balls, and we had to use level 2, it was proposed to get CatBot to lift the black ball on top of the other robot. Achieving this would have been a significant challenge.



	Round1	Time	Field	Round2	Time	Field	Best Run	
STL14	100	08:00	4	30	08:00	2	**100**	**08:00**
STL7	10	08:00	3	90	08:00	1	**90**	**08:00**
STL12	10	08:00	4	80	03:40	2	**80**	**03:40**
STL11	0	02:47	2	70	07:54	4	**70**	**07:54**
STL13	60	05:42	4	10	08:00	2	**60**	**05:42**
STL10	60	08:00	2	10	08:00	4	**60**	**08:00**
STL5	60	08:00	3	40	08:00	1	**60**	**08:00**
STL2	0	03:54	1	20	08:00	3	**20**	**08:00**
STL3	20	04:43	1	20	04:32	3	**20**	**04:32**
STL4	0	07:30	1	20	06:27	3	**20**	**06:27**
STL9	20	08:00	2	20	08:00	4	**20**	**08:00**
STL8	0	01:45	2	0	07:18	4	**0**	**01:45**
STL1	0	07:15	1	0	07:14	3	**0**	**07:14**
STL6	-	-	-	-	-	-	**-**	**-**

***Table 1 –** SuperTeam Results, we are STL13*

***Figure 96 –** Working on the robots*

***Figure 97 –** Starting Round*

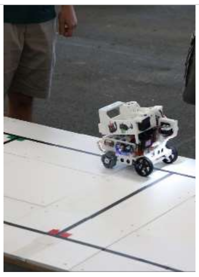


***Figure 98 –** Mid Round*

***Figure 99 –** Apollo & CatBot Teams*

Results:

Final Placing Overall	**10th**
Technical Documents Overall	**2nd**
SuperTeam Overall	**5th**
Poster	**1st** *(100%)*
Engineering Journal	**2nd** (95%)
TDP	**12th** *(73%)*

Conclusions:

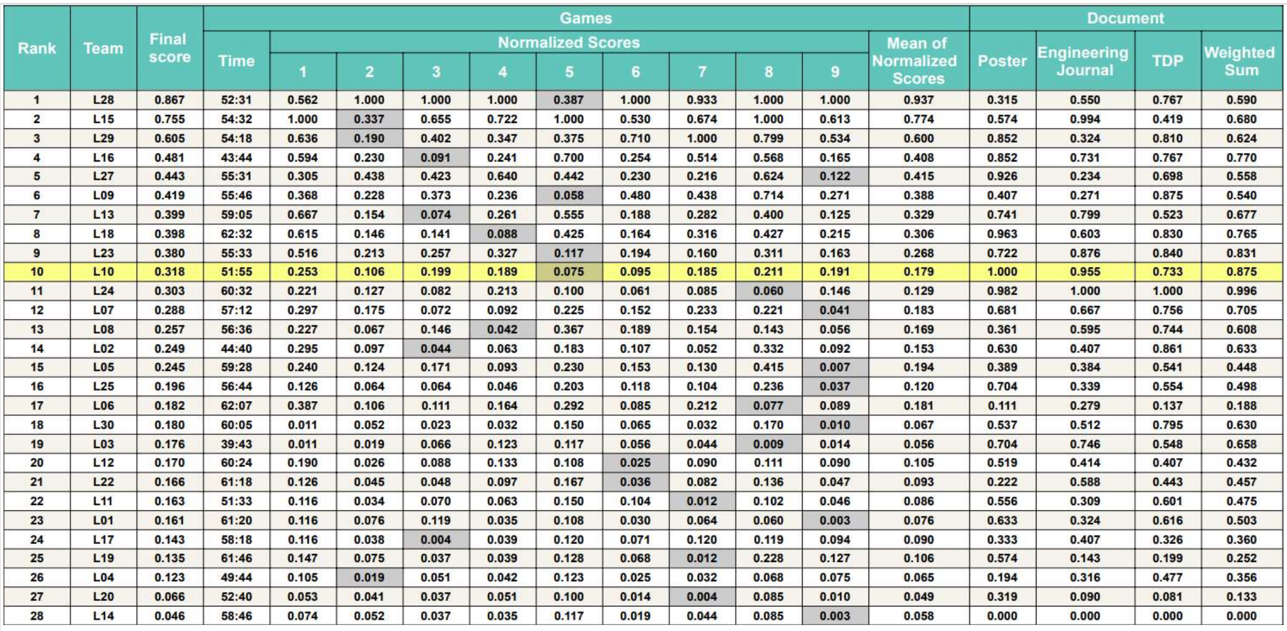
Overall, competing in the RCJI competition this year was a lot of fun and a great learning experience for us, especially in the camera vision areas. Having gone from no design or robot at all less than 3 months ago, we are really happy with our overall result of 10th place, our main goal was to learn more and expand our knowledge, and we achieved that!

***Figure 100 –** Team Australia at RCJI 2023*

If we had the chance to attend a future competition, the major things we'd look at improving would be:

- Processing power, the bot, especially at comp, struggled to achieve over 15-30fps, it'd be good to look at options improve this, perhaps using something other than a pi
- Use a machine learning model to assist with evacuation, the current method of searching for victims is highly problematic. This would also let us learn more about AI. Would likely need a new HW platform
- 28deg ramps, current bot struggles with higher than 26, but the tolerance in the rules allows 27.5
- Debris, CatBot would often get stuck on it during the comp – it was a lot meaner than we do in Aus
- Obstacles, currently can't deal with multi-size, and reconnecting to the line was fairly tough (Fig. 91)

The overall design of the robot worked pretty well, it's definitely capable of solving the whole course, so designing from scratch again would likely not be necessary. It'd be good to build upon this design further.



Rank	Team	Final score	Games											Document			
			Time	Normalized Scores									Mean of Normalized Scores	Poster	Engineering Journal	TDP	Weighted Sum
				1	2	3	4	5	6	7	8	9					
1	L28	0.867	52:31	0.562	1.000	1.000	1.000	0.387	1.000	0.933	1.000	1.000	0.937	0.315	0.550	0.767	0.590
2	L15	0.755	54:32	1.000	0.337	0.655	0.722	1.000	0.530	0.674	1.000	0.613	0.774	0.574	0.994	0.419	0.680
3	L29	0.605	54:18	0.636	0.190	0.402	0.347	0.375	0.710	1.000	0.799	0.534	0.600	0.852	0.324	0.810	0.624
4	L16	0.481	43:44	0.594	0.230	0.091	0.241	0.700	0.254	0.514	0.568	0.165	0.408	0.852	0.731	0.767	0.770
5	L27	0.443	55:31	0.305	0.438	0.423	0.640	0.442	0.230	0.216	0.624	0.122	0.415	0.926	0.234	0.698	0.558
6	L09	0.419	55:46	0.368	0.228	0.373	0.236	0.058	0.480	0.438	0.714	0.271	0.388	0.407	0.271	0.875	0.540
7	L13	0.399	59:05	0.667	0.154	0.074	0.261	0.555	0.188	0.282	0.400	0.125	0.329	0.741	0.799	0.523	0.677
8	L18	0.398	62:32	0.615	0.146	0.141	0.088	0.425	0.164	0.316	0.427	0.215	0.306	0.963	0.603	0.830	0.765
9	L23	0.380	55:33	0.516	0.213	0.257	0.327	0.117	0.194	0.160	0.311	0.163	0.268	0.722	0.876	0.840	0.831
10	L10	0.318	51:55	0.253	0.106	0.199	0.189	0.075	0.095	0.185	0.211	0.191	0.179	1.000	0.955	0.733	0.875
11	L24	0.303	60:32	0.221	0.127	0.082	0.213	0.100	0.061	0.085	0.060	0.146	0.129	0.982	1.000	1.000	0.996
12	L07	0.288	57:12	0.297	0.175	0.072	0.092	0.225	0.152	0.233	0.221	0.041	0.183	0.681	0.667	0.756	0.705
13	L08	0.257	56:36	0.227	0.067	0.146	0.042	0.367	0.189	0.154	0.143	0.056	0.169	0.361	0.595	0.744	0.608
14	L02	0.249	44:40	0.295	0.097	0.044	0.063	0.183	0.107	0.052	0.332	0.092	0.153	0.630	0.407	0.861	0.633
15	L05	0.245	59:28	0.240	0.124	0.171	0.093	0.230	0.153	0.130	0.415	0.007	0.194	0.389	0.384	0.541	0.448
16	L25	0.196	56:44	0.126	0.064	0.064	0.046	0.203	0.118	0.104	0.236	0.037	0.120	0.704	0.339	0.554	0.498
17	L06	0.182	62:07	0.387	0.106	0.111	0.164	0.292	0.085	0.212	0.077	0.089	0.181	0.111	0.279	0.137	0.188
18	L30	0.180	60:05	0.011	0.052	0.023	0.032	0.150	0.065	0.032	0.170	0.010	0.067	0.537	0.512	0.795	0.630
19	L03	0.176	39:43	0.011	0.019	0.066	0.123	0.117	0.056	0.044	0.009	0.014	0.056	0.704	0.746	0.548	0.658
20	L12	0.170	60:24	0.190	0.026	0.088	0.133	0.108	0.025	0.090	0.111	0.090	0.105	0.519	0.414	0.407	0.432
21	L22	0.166	61:18	0.126	0.045	0.048	0.097	0.167	0.036	0.082	0.136	0.047	0.093	0.222	0.588	0.443	0.457
22	L11	0.163	51:33	0.116	0.034	0.070	0.063	0.150	0.104	0.012	0.102	0.046	0.086	0.556	0.309	0.601	0.475
23	L01	0.161	61:20	0.116	0.076	0.119	0.035	0.108	0.030	0.064	0.060	0.003	0.076	0.633	0.324	0.616	0.503
24	L17	0.143	58:18	0.116	0.038	0.004	0.039	0.120	0.071	0.120	0.119	0.094	0.090	0.333	0.407	0.326	0.360
25	L19	0.135	61:46	0.147	0.075	0.037	0.039	0.128	0.068	0.012	0.228	0.127	0.106	0.574	0.143	0.199	0.252
26	L04	0.123	49:44	0.105	0.019	0.051	0.042	0.123	0.025	0.032	0.068	0.075	0.065	0.194	0.316	0.477	0.356
27	L20	0.066	52:40	0.053	0.041	0.037	0.051	0.100	0.014	0.004	0.085	0.010	0.049	0.319	0.090	0.081	0.133
28	L14	0.046	58:46	0.074	0.052	0.037	0.035	0.117	0.019	0.044	0.085	0.003	0.058	0.000	0.000	0.000	0.000

***Table 1 –** Overall Final Results, CatBot (L10) Highlighted*