RoboCupJunior Rescue Line 2023

Team Description Paper

Bugless

Abstract

This paper presents the development and evaluation of a camera vision-based rescue line robot designed for optimal performance for rescue league. Departing from conventional light sensor line tracking, our robot employs camera vision to navigate the rescue line accurately. The integration of custom-made, 3D printed PLA parts ensures seamless compatibility with the software, enhancing overall functionality and efficiency. Through rigorous testing and analysis, we evaluate the performance of our robot, considering factors such as line detection accuracy, obstacle avoidance capability, and manoeuvring efficiency. The results demonstrate the effectiveness of our approach in achieving reliable and precise line tracking for rescue operations. Ultimately, this research will be able to contribute to the field of rescue robotics, possibly offering insights into the integration of camera vision and custom-made components to enhance the performance of autonomous robots in real life scenarios.

1. Introduction

a. Team

Our team consists of four members, Celeste Tan, Lim Kae Sophie, Xue Minjie and Aseera Jannath. Celeste Tan is the team leader, who is adept in both software and hardware, integrating both aspects into the project while ensuring coordination among team members. Lim Kae Sophie is the hardware specialist, who has contributed her profound technical expertise in handling intricate hardware components, ensuring seamless compatibility and reliability in our robot. Xue Minjie is our software specialist, who has contributed to the fluid integration of software components, while also ensuring optimal performance and functionality of our robot. Aseera Jannath also excels in software development, with their acute attention to detail being a huge asset to our team, ensuring reliability of our robot.

2. Project Planning

a. Project Plan

Aims for the competition

Through this competition, we hope to explore different methods and techniques of approaching robotics to design and develop an efficient and reliable autonomous robot, while coming up with innovative solutions to the various challenges in the competition. We hope to continually improve our robot based on past performances and aim to enhance the robustness and reliability of our robot.

Milestones/Timeline

- Research on innovative mechanisms: This was one of our first steps where we were trying to optimise the hardware for optimal performance of the robot. We researched some mechanisms and gained inspiration from other people's works. It was also a time for reflection on older versions of the robot to identify areas for improvement
- Planning bot structure: We considered software and hardware aspects in relation to each other to plan out the entire robot structure. In January, we informally arranged the parts on the robot in Fusion360 as a rough draft of how the robot would be (without any specifics such as individual components on the pi or motor driver, the parts of our robot were represented by simple boxes). We designed a CAD of our robot in Fusion360 to plan any

major hardware or structural changes before implementing them.

- Timeline planning and task delegation: The next step was to plan out the timeline and set small milestones for us to evaluate our progress. We also delegated tasks amongst members to get different aspects of the bot working.
  - Main milestones: January - Settled design of and ordered PCB, February - Basic code and line track ready, March - Full linetrack and evac zone code, April: Full linetrack and evac zone code together, able to run a full run reliably.
  - Task delegation: Hardware (Kae) does 3d printing for mounts, Software is delegated to different members with each member taking a part of the code (e.g. Minjie on rescue kit, Aseera on obstacle, Celeste on basic line track). PCB is prepared by all members of the team on an online call with screen share. If anyone finished their task early, they will be delegated to help someone else who is having trouble, or work on the next item on the list.
- Testing and improvement: A huge part of our time was invested in this area as we had to test how well our ideas worked out in reality and look for improvements to be made. We made multiple versions of all hardware components (PCB, 3D prints) to improve our robot's capabilities after extensive testing.
- An important aspect of our whole project is also taking inspiration from others, not just learning about what worked but also what didn't so that we do not repeat the same mistakes

Constraints

- By taking into account the requirements of the task, we needed to allocate more time for the actual planning of the robot layout and structures as this would be essential for the rest of the plan to be carried out.
- The size of our robot is limited by the size and arrangement of our components, as well as the limited precision of 3D printing at a small scale
- Due to the limited budget provided to us, we had to carefully nitpick the components to use on our bot, minding cost.

b. Integration plan

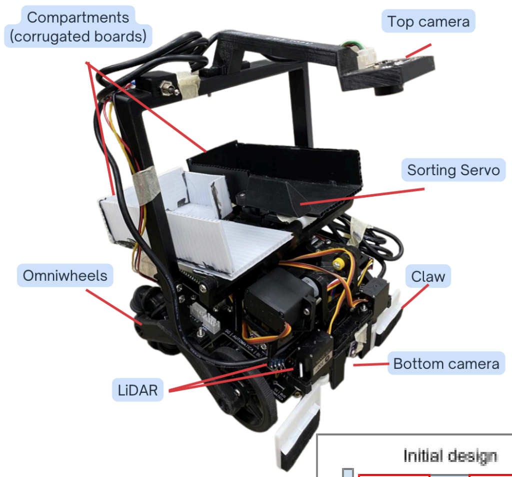
- For efficient coordination between team members, we schedule regular meetups, and make use of platforms such as Github, VS Code (Live sharing) and Fusion360 that allow real-time collaboration and/or help us maintain an updated common repository.
- We have a centralised meetup area that houses all our tools and components to ensure everything can be easily retrieved whenever we need it.
- All parts of our robot are constantly being evaluated; we note down areas of improvement for mechanical, software and electrical aspects of the bot, and strive to create better iterations of our bot every time we update parts of the robot.
- We tested all available options we have, such as trying to use light sensors, bottom camera and top camera for line track, and trying out different types of light sensors until we found one that best suited our needs.
- After thorough and careful research, we have carefully selected and integrated various components, including motors, LiDARs, servos, cameras and microcontrollers, for reasons elaborated below in this entire document.

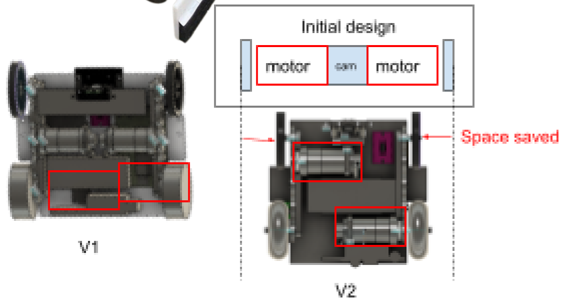
Hardware

Main Structure

These are the main features of our robot. The 6 LiDARs and 2 cameras are our main inputs, providing us with information about the robot's movement. It is driven by 2 DC motors connected to the drive base consisting of 2 rear omni wheels and 2 normal front wheels. Our rescue mechanism consists of the gripper, which grabs the rescue kit and victims and lifts them into the compartments. The sorter turns based on the object collected to sort the rescue kit and alive victims into one compartment and the dead victims into the other for deposition.

a. Mechanical Design and Manufacturing





Drivebase

The robot is driven through differential steer, and has four wheels in total with the left and right wheels independently driven with one motor for each side. The front and back wheels on each side are connected through a gear train.

Space Constraints

We used encoded motors in order to implement PID in our driving. Encoded motors allow the robot to cross speed bumps and ramps that it would otherwise stall at. With encoders, PID can be used to account for the error in motor speed needed, giving accurate motor control.

However, this increases the length of the motors, increasing the minimum width of the robot greatly at the front, since the camera is placed between the motors to remain at the pivot point. Initially, V1 had the motors placed side by side but away from the camera. V2 places the motors at an offset so that the width of the robot can be reduced more. This is especially valuable for tolerance when entering and exiting the evacuation zone and when navigating around obstacles as there might be a column, ramp or wall nearby, making it a tight fit.

Gearbox

The use of a gear train and gear box gives us more options and allows us to offset our motors to optimise space and minimise the size of our robot. Furthermore, both wheels on one side can be driven by the same motor, saving space compared to driving each wheel with a separate motor.

Gearbox Development

We went through multiple rounds of testing where we printed out small test prints modularly and incorporated the best working solution of each aspect into the current gearbox mechanism.

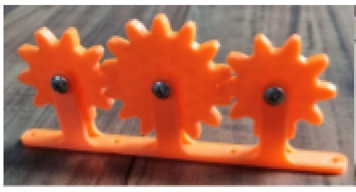


Fig 1.1

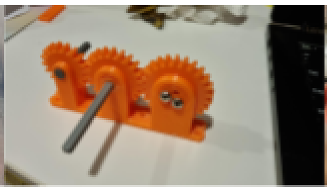


Fig 1.2

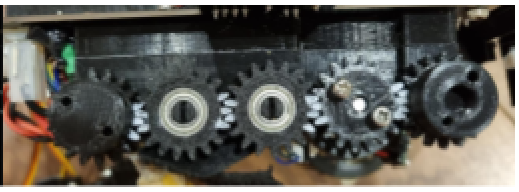
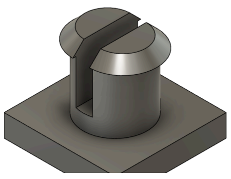


Fig 1.3

The first prototype (Fig 1.1) involved using screws as pivot points but significant friction resulted in jerky movements. The second prototype (Fig 1.2) used lego axles as pivots, but the gears were constantly shaky and shifting in other axes.
The final design (Fig 1.3) utilised ball bearings as they resulted in the smoothest movement. To prevent the gears from slipping off the ball bearings, they were secured using epoxy glue, when our final design was confirmed right before the nationals.

Snap-Fit Mechanism

The ball bearings are attached to the gearbox using a snap-fit mechanism, and can be removed by squeezing the two prongs together and pulling the ball bearing and gear out. We chose this mounting option over glueing the ball bearing directly to the



gearbox as it allows us to swap gears out more easily
The design is 3D printed in a sideways orientation such that the layer lines are not along the stress points created when bending the arms.

Omni Wheels

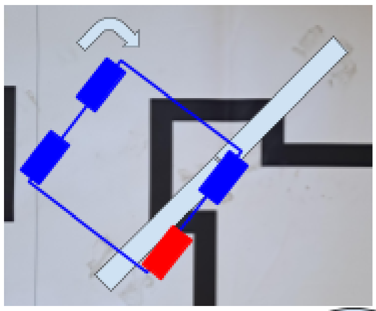
The omni wheels allow the back of the robot to translate perpendicularly to the robot's direction of travel, especially when turning, so that the pivot point of the robot can be at the front, slightly behind the middle of the front wheel axis. This ensures consistency in movement, so even when the centre of gravity changes (e.g. when moving backwards), the pivot point remains the same.

Custom-Made Omni Wheels Structure

Our omni wheels are made by placing 20 mm diameter rubber o-rings around 3d printed donuts as seen below. Metal dowels are slotted through these prints and placed in one half of the omni wheel housing which has depressions for the metal dowels to rest upon, and the second half is then screwed onto the first to complete the omni wheel. The total diameter of the wheel is 60mm.

Accurate Odometry

Due to the difference in diameter of commercially available omni wheels and our drive wheel, with one full rotation of the drive wheel, the omni wheel rotates more than one full rotation. Over time, this causes the omni wheels to slip, and the rubber rollers will slip across the floor causing inaccurate odometry and unreliable positioning of the robot. The 3d printed omni wheel is designed to match the diameter of our front wheel, preventing slipping when rotated at the same angular velocity. Furthermore, the o-rings were specifically chosen to help overcome other parts of the challenge, such as speed bumps.



Speed Bumps

All commercially obtained omni wheels have rollers of a diameter smaller than 20mm (Figure 2.1). When the omni wheels need to cross speed bumps laterally, such as on a sharp turn, it requires a strong sideways frictional force to cross the speed bump. The robot was unable to get across the speed bump in this situation, even with rigorous tuning, and the robot would get stuck. The 10mm radius o-rings (Fig 2.2, 2.3) make it easier for the robot to cross the speed bump.

Fig 2.1

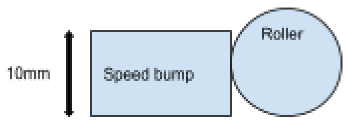


Fig 2.2

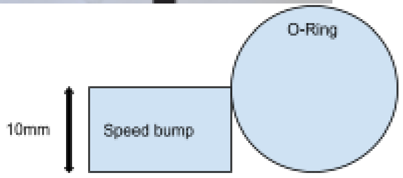
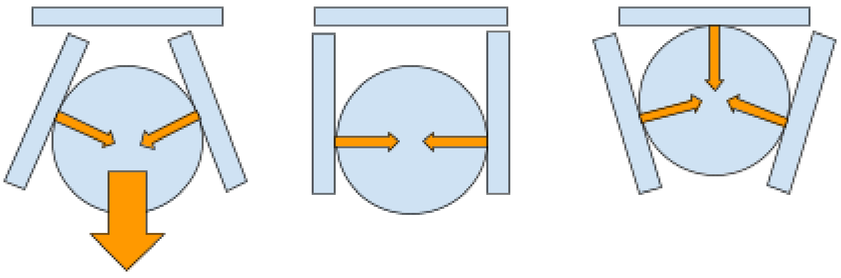


Fig 2.3

Rescue Mechanism

Our rescue mechanism has 3 main hardware elements, the gripper, the sorter and the 2 compartments. The gripper picks up the rescue kit, and the balls in the evacuation zone; the sorter sorts the balls according to their colour as detected by the top camera. Our compartments store the kit, and the live and dead victims separately for deposition at the evacuation points, allowing us to pick up victims in any order.



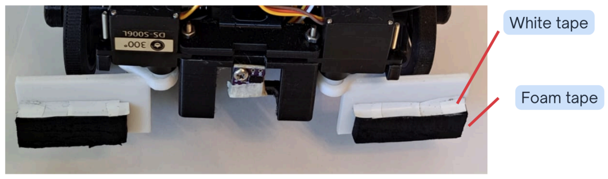
Gripper

Our white flat grippers are covered with foam tape, suspended more than 10mm from the ground to allow it to pass over speed bumps.

Distance Between Grippers

When the gripper hits the ball at an angle as the distance between the two grippers is less than 50mm, the ball sometimes ‘rockets’ away due to the force exerted away from the robot (see diagram above). To solve this, the distance between the two grippers, when parallel, is the diameter of the larger ball, 50mm. This enables the gripper to grip smaller balls in a 3-point grip and 50mm balls in a 2-point grip, keeping the ball within the grippers.

Flat Gripper



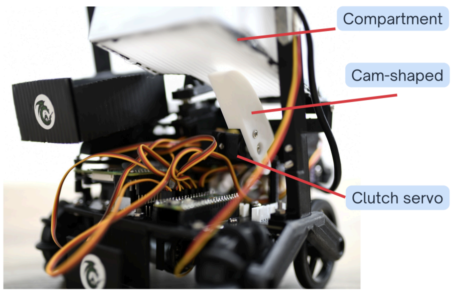
Initially, the robot was unable to be flush to the deposit point due to a curved gripper. With the flat gripper design, the robot is able to be flush against the deposit point (gripper flat against the deposit points) and prevent any objects from falling short of the deposit point when it deposits.

Foam Tape

The plastic grippers are too smooth to grip the ball tightly on their own, so we used a foam tape, which was soft and conforms to the shape of the balls to increase the surface area of contact.

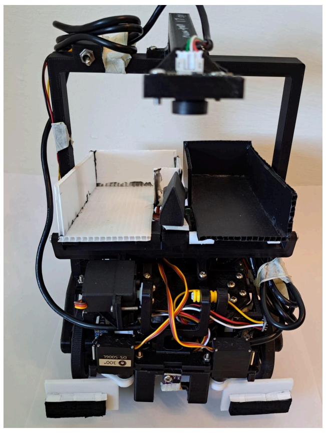
Colour of Gripper

Our initial design used black coloured grippers. However, this was problematic as we used our top camera to differentiate between live and dead victims, based on the percentage of black pixels in the masked out image of the balls. Since both the gripper and foam were black, part of the gripper would contribute to the percentage of black in the mask, causing the camera to assume a larger percentage of black, which could result in silver balls being falsely detected as black. Thus, our subsequent grippers were printed in white, and white tape covers the black foam at the top (as seen in the image above) so that when viewed from the top camera, the only source of black would be from the balls.



Compartments

There are two compartments, a larger one for ‘live’ victims and a smaller one for ‘dead’ victims. They are hinged at the front of the robot, and lifted from the back with individual servos through a cam. This allows for more efficient and precise movement as the robot does not need to reposition itself after detecting the deposit point.



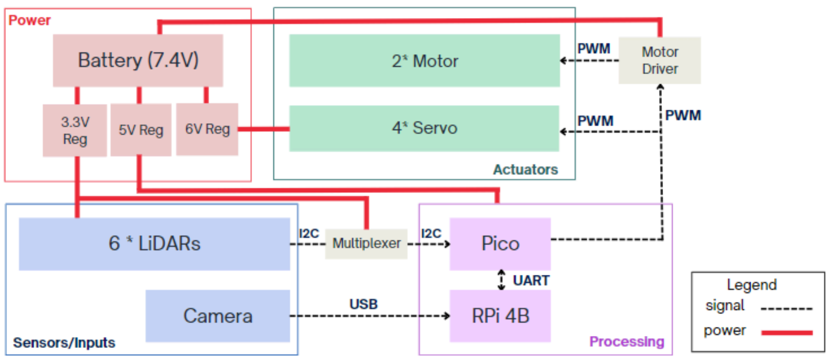
Individual Compartments

The two compartments ensure that the two deposits are independent, and the victims will not be accidentally deposited at the wrong evacuation point Furthermore, victims can be collected in any order as order of deposit is not affected by order of collection, as they can always be sorted into their respective compartments.

b. Electronic Design and Manufacturing

Overview and Electronics Diagram

Electronics Diagram



Above is a brief overview of our electronics design. We step down the voltage from the battery using 3 different voltage regulators to supply power to the various components. Most of the components above are connected through our custom printed circuit board that also serves as the chassis of the robot.

Printed Circuit Board

We designed our own printed circuit boards to improve the organisation of all the components on our bot. Here are some of our considerations when designing the PCB.

- Copper Plane: In our PCB design, we opted to use copper pouring to optimise the power supply distribution of the robot. Copper planes for 3.3V, 5V, 6V, 7.4V and GND are used to quickly connect all the necessary pins without worrying about running out of space for traces. Furthermore, copper's thermal conductivity also allows the plane to act as a heat sink to help dissipate heat and maintain low operating temperatures. We generally grouped components with similar voltage requirements together on the PCB so they can be easily connected through a copper plane.
- Trace widths: We calculated the minimum trace width required for all the connections on our PCB to prevent overheating. This is especially important for traces delivering large currents. Since the heating effect of a wire is given by $P = I^2R$, and the resistance of a trace is inversely proportional to the cross-sectional area of the traces (Ohm's Law), these traces are much more likely to overheat and require a larger trace width to prevent it.
- Spare pins: We also broke out spare unused pins of the microcontroller on the PCB in case we add more components.

Sensors and components

Controllers

We used the Raspberry Pi Pico as our microcontroller for its compact size, affordability and versatility. Compared to alternatives like Arduino Mega, Raspberry Pi Pico offers a cost-effective solution without compromising performance, with its high amount of IO. It has 2 I2C buses, 2 hardware serial, and 4 software serial enabled by PIO. The microcontroller's efficient processing power and real-time control capabilities enable us to manage and coordinate our actuators, as well as acquire and process data from various sensors.

We use the Raspberry Pi 4B in conjunction with OpenCV and Python for image processing. On the pi, the user has no control over when a given process is executed, because the order in which processes are executed is determined by a scheduler, with no set time limit for completion. On Pico, the user can utilise timing and interrupts to ensure that sensor reading and motor control operations occur within a tight timing budget.

Clutch Servos (3x 300° DS-S006L (9g), 3x 300° DS-R005 (2kg))

We use clutch servos for the gripper, deposit servo and sorting servo. In conventional servos, stalling occurs when the load applied exceeds the torque capability of the servo motor. When a normal servo is stalled for an extended period, the motor experiences increased current draw, which generates excess heat. This heat accumulation can potentially lead to thermal damage and eventual burnout of the servos. However, clutch servos help to mitigate this overheating problem by disengaging clutch mechanisms of the motor when it is about to stall, which prevents sudden shifts in power and hence prevents excessive power draw and overheating.

Motor driver (DRV8833 Dual H-Bridge Motor Driver)

We use 2 DRV8833 motor drivers which are surface-mounted onto our PCB to save space. To get a 3A continuous current which is what our motor needs (3.2A max current with no load), we used a HTSSOP package with a thermal pad that allows for greater heat dissipation and thus greater current (3-A RMS, 4-A Peak), as compared to the TSSOP package without a thermal pad (1-A RMS, 4-A Peak), and we routed the input and outputs in parallel as shown:

LiDARs (5X VL53L0X and 1X VL53L1X)

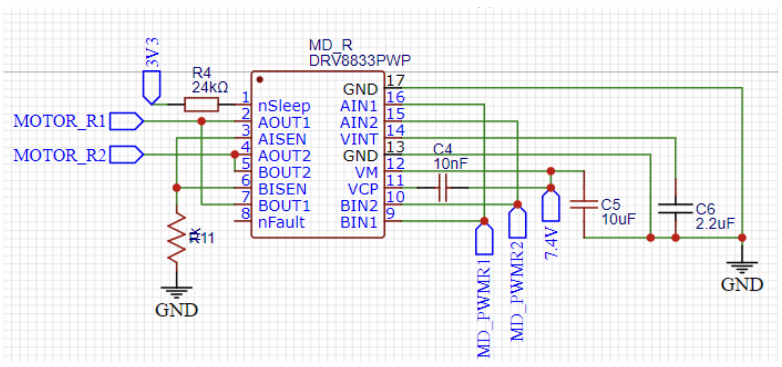
We used VL53L0X and VL53L1X, ToF sensors, as our distance sensors as it has a relatively small cone of sensing (25°) and allows us to detect long-range distances accurately. This is useful when we are using them to track the walls in the evacuation zone where the walls can sometimes be a significant distance away from the sensors. We chose this over the ultrasonic sensors as its larger cone of sensing makes long-distance sensing less reliable as more objects interfered with the readings.

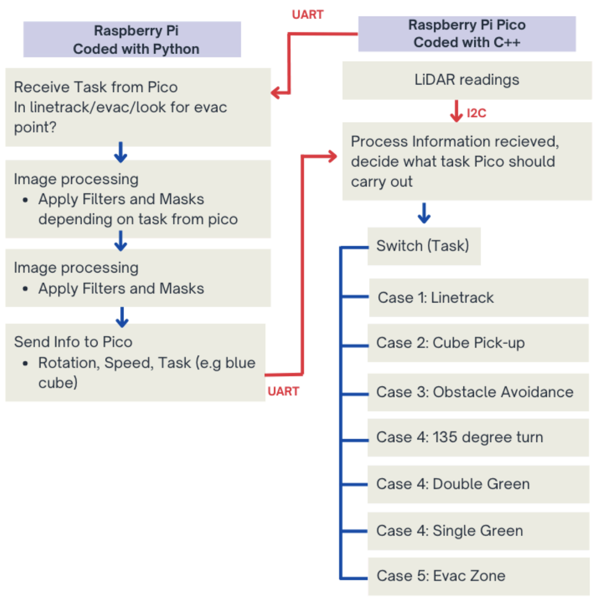
<u>RGB Light Sensors (6X TCS3472)</u>

We mount light sensors on a separate PCB at the front of our robot to assist in line track and silver line detection.

<u>Multiplexer</u>

We are using the TCA9548A I2C multiplexer so that we can read from all the six LiDARs and all 6 light sensors in the same loop. Since they are all of the same address, 0x29, readdressing is required to be able to read from all of them, but there aren't enough pins on the pico alone to readdress them all. The multiplexer is surface mounted onto our PCB as well to save space.





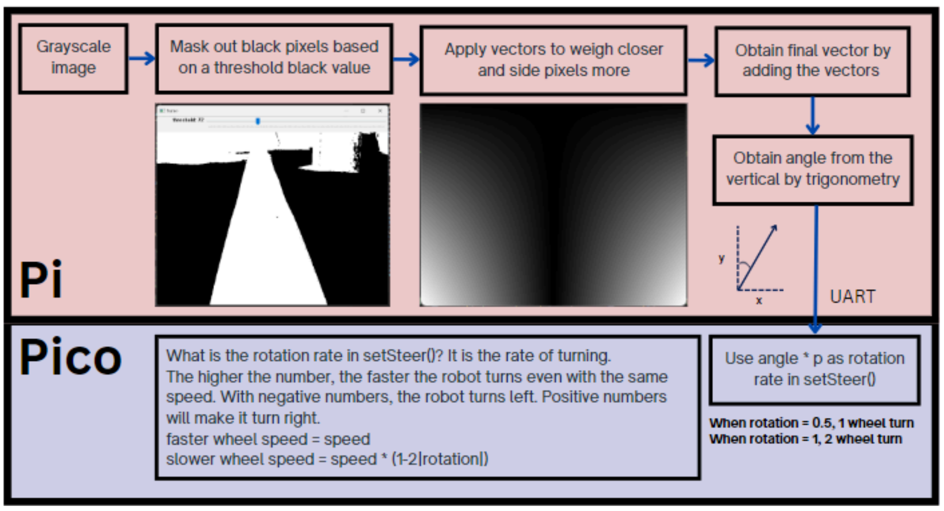
Software

Main Loop

Figure X is a high level overview of our main loop in the bot. We use the Pico to control our actuators and the robot's movement using a state machine, with each case referring to tasks that the bot has to complete. The Raspberry Pi communicates with the Pico and sends information such as the speed, rotation and task to the Pico based on what it sees after processing the images from the camera. The Pico also sends information such as whether it's currently line tracking or in the evacuation zone so that the Pi can apply the right image filters and send the right information to the Pico.

Colour Space

We use the HSV color space instead of the RGB space for image processing. The HSV space proves more advantageous as it allows for effective color masking by thresholding using the hue channel. RGB values are also highly sensitive to variations in lighting conditions, making it even more challenging to accurately precisely determine colors.


Grayscale image
Mask out black pixels based on a threshold black value
Apply vectors to weigh closer and side pixels more
Obtain final vector by adding the vectors
Obtain angle from the vertical by trigonometry
Pi
y
x
UART
Pico
Use angle * p as rotation rate in setSteer()
When rotation = 0.5, 1 wheel turn
When rotation = 1, 2 wheel turn

What is the rotation rate in setSteer()? It is the rate of turning. The higher the number, the faster the robot turns even with the same speed. With negative numbers, the robot turns left. Positive numbers will make it turn right.
faster wheel speed = speed
slower wheel speed = speed * (1-2|rotation|)

Line Track

We implemented an innovative approach for line tracking utilizing a pre-populated vector matrix, which accounts for pixel distance from the origin. The weighted sum of the matrix yields the overall output vector, which determines the target angle and rotation value (See below). Thus we can prioritise the lines that are closer to the robot, as seen in the "final_x" window where the sides that are nearer to the bottom are whiter (or higher in value) than the further or closer ones.

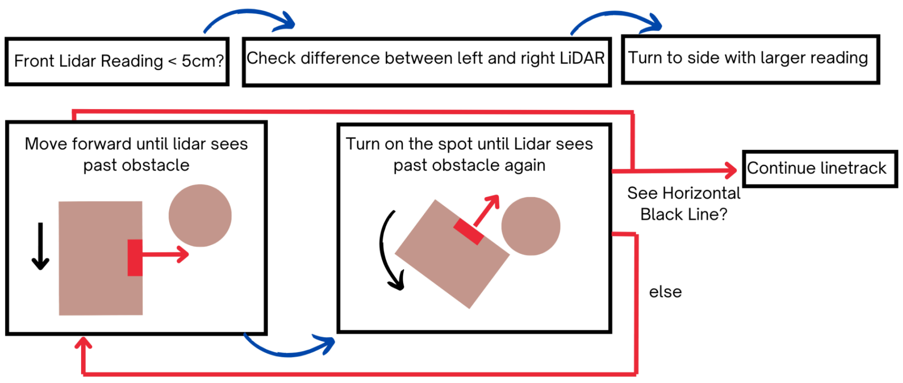
Green Squares

To determine the direction to turn when encountering green squares, both the positional relationship between green squares and black lines, as well as the number of green squares present have to be considered.

Firstly, the sum of green pixels in the bottom camera frame is computed. If there are sufficient green pixels, we check the position of black lines and percentage of black in relation to the green squares. The relative positions of black pixels in the frame will determine whether the green squares will be taken into consideration. For instance, if the green squares lie above the black line, they will be ignored. Otherwise, the centroid of the black region is then computed, from which we can derive whether the green squares are to the left or right of the line segment, or if there are double squares, and execute the correct turn.

Obstacle Avoidance

We tried out various methods before coming up with the algorithm as seen below:


Front Lidar Reading < 5cm?
Check difference between left and right LiDAR
Turn to side with larger reading
Move forward until lidar sees past obstacle
Turn on the spot until Lidar sees past obstacle again
Continue linetrack
See Horizontal Black Line?
else

First, the robot checks which side to turn to by comparing the left and right side LiDAR readings. The robot will perform the above 2 main actions (see diagram) in a loop which causes it to move around the obstacle until it sees a line. Refer to the appendix for other methods we tried previously.

Rescue Kit

The collection of the rescue kit and successful return of the bot to the line relies on a combination of odometry and strategic positioning.

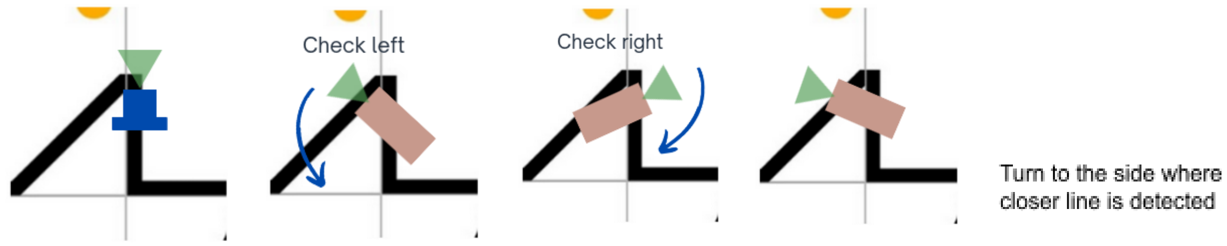
To detect the rescue kit, the sum of blue pixels in the bottom camera frame is computed. If there are sufficient blue pixels, the robot will start rotating towards the blue cube. To ensure that the bot moves directly towards the rescue kit, the camera is aligned with the centroid of the kit before moving forward, which allows for more accurate positioning. This helps to optimise its trajectory, and ensure that the robot will pick up the cube reliably. We also decided to only approach the cube once the bot is sufficiently close to it so that the robot can return to the line relatively quicker.

Once the camera is aligned with the kit's centroid, the robot will drive forward. During both the rotational and forward motion, the distances travelled by each motor are recorded through the encoder values. This enables the robot to effectively trace its rotational and forward displacements. Upon reaching the cube, as indicated by our front LiDARs, the bot will pick it up. Then, it will navigate back to the line, by computing displacement from its last known line via the encoder values mentioned above.

Line Gap and 135-degree Turns

When encountering an end of a line, the bot first performs 90-degree sweeps in both directions on the spot to identify potential 90 and 135-degree turns. Utilising the bottom camera, the bot keeps track of the closest line detected on either side and turns in that direction.

If no line is detected, the bot performs a straight drive from its original orientation until it detects the next line segment. As it moves, the bot scans within an inverted cone-shaped region to account for the perspective of the bottom camera (Fig Yy). This also helps to account for potential errors in the initial heading of the robot; even if the robot's initial angle was off, the top side of the cone would ensure that the continuation of the line following the line gap remains captured within the mask. This is advantageous compared to a rectangular mask, as the latter may not be able to capture the line continuation adequately. Further pixels are also given more weight than nearer pixels. This is to ensure that the bot will not latch on to other adjacent lines, and will drive straight. However, there were some problems that we encountered using this approach with the bottom camera, which is mentioned in the appendix.



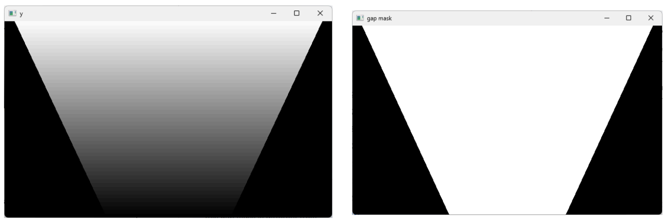


Figure Yy

Evacuation Zone

We adopted a spiralling approach to navigate the evacuation (evac) zone, utilising diagonal LiDARs. The evac zone consists of three main sections: wall-track and ball collection, deposition, and exit.

Wall-Track and Ball Collection

Using LiDAR measurements, we used proportional control to maintain a certain distance from the wall, known as the setpoint distance. The larger the error, the larger the steering rate for the bot to correct itself until it is at the ideal setpoint distance. The distance from the wall is a function of time (as time passes, the setpoint distance increases), allowing us to accomplish a spiralling motion.

We also used the Hough circles function from OpenCV to detect the balls. The top camera is used instead of the bottom camera so that the ball cannot block the entire FOV of the camera. The bot will move towards any balls it identifies while spiralling. To differentiate between ball types, the percentage of black pixels in the masked out balls is calculated. Once the balls are close enough to the grippers as indicated by the front LiDAR readings, the bot will initiate a pickup sequence. The balls will then subsequently be sorted into their respective components.

Deposition

After spiralling into the centre of the evac zone, the bot will spin in place to detect the two deposit points corresponding to the alive and dead deposit points. The bot will then travel towards the centroid of the deposit points and initiate the deposit sequence. The bot will then travel back to the centre to locate the other deposit point and repeat the same process.

Exit

Once finished depositing, the bot will return to walltracking to try and find the exit, which is indicated by an opening in the evac zone with a black strip. The openings are detected by a drastic jump in the diagonal LiDAR readings (eg. out of range), and the black strip is detected using the top camera to confirm that it is the exit instead of the entrance (marked by a silver tape instead). On successful exit, the bot will then transition back into linetrack.

Performance Evaluation

Line Track Tests

We first conducted tests on individual tiles with various line patterns. then moved on to use combinations of tiles, to account for cases when the robot's initial position when entering a tile is not ideal. Finally we validated our bot with complete competition mats from 2022 & 2023 SG Open and 2022 Virtual RCAP, which contain challenging obstacles such as narrow turns with speed bumps, and lines on separate tiles that are situated close to one another.

The tests involved measuring the accuracy of the robot's trajectory, and its ability to maintain a consistent path while tracking the line by checking its reaction time to correct itself during sharp turns, curves and linegaps. We recorded the deviation from the desired path and analysed the data to assess the precision and reliability of the line track. By collating data on the types of tiles that the robot does not perform well for, we fine tuned our bot accordingly.

In order to evaluate the rescue kit detection functionality, we set up scenarios with varying cube positions and orientations. The robot was tasked with detecting and accurately approaching the cubes. We measured the success rate of cube detection and analysed the data to determine the robot's proficiency in identifying and locating the cubes within the given environments.

Hardware tests

We started off with unit testing, followed by sub-system and system tests. Using the LiDARs as an example, individual testing was conducted by connecting each LiDAR separately to the microcontroller. Subsequently, the accuracy of the LiDARs was evaluated by comparing a known distance with the reading of the LiDAR. The LiDARs were then integrated with other aspects of the code such as line track to assess their impact on the overall robot performance. Any potential interference or issues were swiftly addressed through the debugging process. Once the integration was successful, the LiDARs were incorporated into the main codebase.

Similar testing and integration procedures were conducted for all other components. This comprehensive testing ensures the reliability and functionality of our robot, contributing to its overall performance and success in fulfilling its intended tasks. For 3D prints, we start off with unit testing as well where we print tests for individual mechanisms. For example, for the gearbox, we initially printed the snap-fit mechanism in isolation, assessing its durability and reliability. Subsequently, the mechanism was used in the gearbox, and its overall robustness was examined prior to final integration. Next, we tested the hardware alongside the complete robot configuration to identify and address any potential issues. This iterative testing process allowed for continuous refinement of our design, ensuring accuracy and reliability in fulfilling its intended purpose.

The test results played a crucial role in the development process. providing valuable feedback on the robot's strengths and areas for improvement, allowing us to make informed decisions on refining the algorithms, adjusting sensor parameters, and optimising control mechanisms. The analysis of the results helped us identify specific challenges and potential limitations of the robot's performance, which informed subsequent iterations of our development strategy.

Conclusion

In conclusion, our research endeavours focused on the development and optimization of a highly capable rescue line robot. Through the utilisation of advanced technologies and innovative design principles, we have successfully addressed key challenges and achieved significant progress in the field of autonomous robotics.

Appendix

<u>Problem with linegap: Bot may be unable to differentiate between tiny gaps between tiles and actual line gaps.</u>

The bot may sometimes treat gaps between tiles as an end of line, resulting in it performing the entire linegap sequence, wasting precious time especially if it makes this mistake many times. In the worst case scenario, the bot may end up drifting from the original line entirely. This gap in tiles issue was a major problem we faced during the RoboCup nationals competition. Hence, this time around, we decided to resolve this issue by using the top camera instead, where the gap between the tiles would not be as significant.

<u>Obstacle: Testing and Previous Ideas</u>

Our previous approach involved using a PID algorithm to navigate around obstacles while maintaining a given set-point distance (Figure 1). The algorithm aimed to minimise the error between the desired distance and the measured distance from the obstacle using the LiDARs.

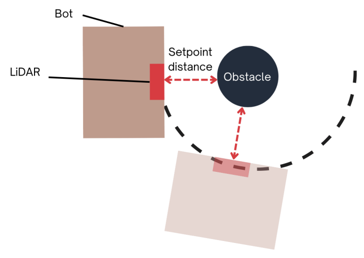


Figure 1

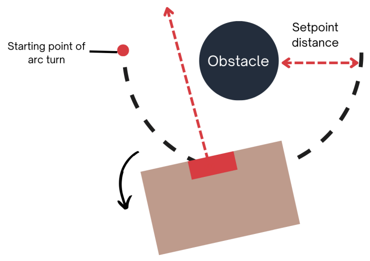


Figure 2

However, we encountered reliability issues due to the limitations of the LiDAR sensor in accurately measuring the distance and accounting for the obstacle's angle. The sensor primarily measures the distance to the obstacle but struggles to accurately determine its angle or orientation. When the LiDAR measures distances beyond the obstacle, such as in the scenario in the diagram on the right, the error would be excessively large, causing the robot to deviate from its intended path and continuously rotate around a fixed point without returning to the desired trajectory. These limitations in accurately computing the error when the obstacle was not reliably detected made it challenging to achieve robust tuning in the PID algorithm. As such, we decided to choose another method that is less susceptible to tuning issues.