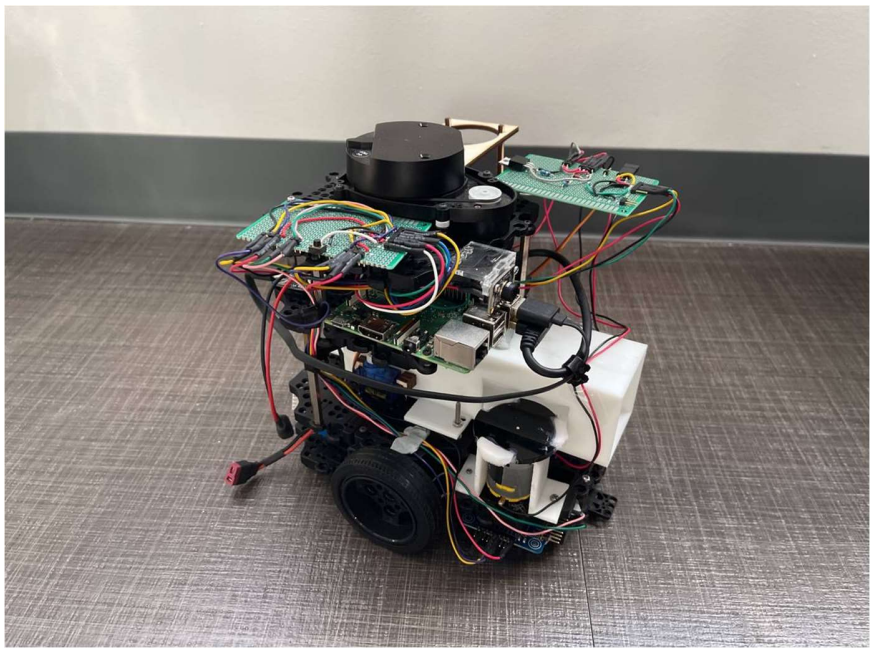
EG2310 Final Report (Group 6)

Aditya Satish Nalini **A0244516H**
Dawn Lim Kia Hwee **A0244935X**
Ethan Yidong Tang **A0244841E**
Josiah Chua En Zhi **A0238950X**
Zann Tan Zhi Ann **A0234755Y**

Design Review



Problem Definition

We will refer to the system collectively as “the turtlebot”.

1. The turtlebot robot is to navigate a maze of not bigger than 5m by 5m. With wall heights under 1m.
    a. The turtlebot should be able to move across a flat, horizontal plane with limited space.
    b. The turtlebot should be able to visualize the layout of the maze surrounding it without any prior knowledge of the maze.

    c. The turtlebot should be able to navigate through a 5m by 5m completely enclosed maze with no dead ends.
2. The robot is to autonomously navigate to a designated loading zone which is identified by a NFC tag on the floor. There are two such zones, and both of which are found along the walls of the maze. Here the turtlebot needs to wait in position to receive up to QTY:05 ping pong balls manually loaded by the TA.
    a. The turtlebot should be able to read the NFC tag, located on the ground.
    b. The turtlebot should be able to accept and load at least one and at most five ping pong balls.
    c. The turtlebot should be able to detect when it is fully loaded, OR the turtlebot should accept a "simple" input that signals that the turtlebot is fully loaded.
        i. "Simple" is not a technical term, nor is it well-defined. To be resolved by asking for the operator's discretion.
3. The robot is to navigate the maze to locate tin cans on the floor ("target"). One out of the 3 targets have an infrared heat signature ("hot target").
    a. The turtlebot should be able to navigate to a location within firing distance of the hot target.
4. The robot is to identify the hot target, aim and fire a minimum QTY:01 ping pong ball at the hot target.
    a. The turtlebot should be able to detect the hot target, that is, the hottest target of the 3 targets.
    b. The turtlebot should be able to aim the firing mechanism for the ping pong ball, AND move to a position such that the firing will hit the hot target
    c. The turtlebot should be able to shoot a ping pong ball such that it will hit the target with appreciable velocity.
    d. The turtlebot should be able to hit the target within five shots.
        i. There is no defined minimum reliability for the firing mechanism, however the reliability should be maximized.

Timing: Not more than 30 minutes
Maintenance: There are no defined maintenance requirements for the turtlebot.

Literature Review

The turtlebot's goal is not much different to that of previous years, save for a few changes in the identification of ping pong ball loading zones and the type and identification of the targets to launch the balls at. It still needs to collect the balls at a designated loading zone and launch them at specific targets, all the while so moving autonomously. Most other requirements for the design still remain the same. This webpage (https://cde.nus.edu.sg/idp/academics/modules/eg2310/) contains videos of previous solutions to this problem.

We were also able to view code and some documentation from the previous years' projects by looking through the forks of shihchengyen/r2auto_nav. In particular, hjunleon/r2auto_nav and alvynben/r2auto_nav provided very detailed documentation on their mounting layouts and components. We noted that both utilized an AMG8833 for IR detection. Both also utilized a flywheel firing mechanism.

Navigation

There are many different algorithms for navigating a maze. The ones we found or thought of are:

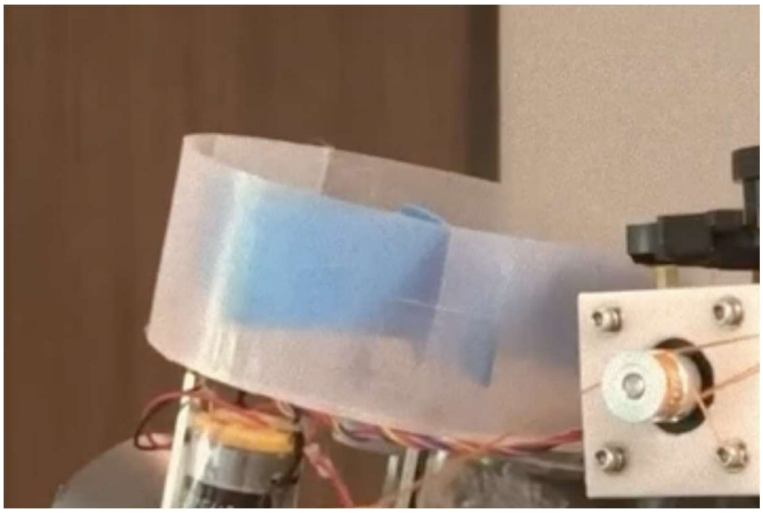
Algorithm	**Description**	**Advantages**	**Disadvantages**
Random (Arya)	At any point in time, if the turtlebot reaches an intersection, the turtlebot will randomly choose between going straight, turning left, or turning right.	Simple to code	Not guaranteed to explore the maze or locate the key locations in a fast manner.
Greedy (r2auto_nav) (Srichandan)	At any point in time, if the turtlebot cannot continue to move forward, it will greedily choose the path which has the longest distance to the wall, and move in that direction.	Simple to code Restriction of "no U-Turns" means we can rule out the situation where the turtlebot just moves back and forth in a straight line by ignoring the back angles.	May get stuck in a loop where the turtlebot will travel in a rectangular path
Follow wall (Arya)	Choose a side (let's say right side). Follow the right wall by making	Not difficult to code	Will get stuck in a loop before exploring the full maze when

	only right turns, unless a right turn is impossible.		there are obstacles that are not attached to the outer walls. Will not explore locations further from the walls.
Exploration	Using the occupancy map and SLAM data, keep track of locations the turtlebot has visited. Use a secondary algorithm (A*, BFS) to navigate the turtlebot towards the closest unvisited locations.	Guaranteed to reliably explore the full maze.	Harder to code.
Exploration Hybrid	Using the exploration idea, but combined with an intermittent technique between choosing the next unvisited location. For example, an exploration/follow wall hybrid algorithm would follow the wall until it reaches a location already visited, then it would navigate to an unvisited wall and try again.	Guaranteed to reliably explore the full maze. More efficient than exploration, as the exploration method may cause the turtlebot to move in suboptimal paths (for example, to cover a spot that it missed)	Harder to code.

Loading and Firing

The main mechanical subsystem to be designed by us is the loading and firing mechanism. Researching different types of ping pong ball launchers yielded a few distinct patterns of mechanism for loading and firing.

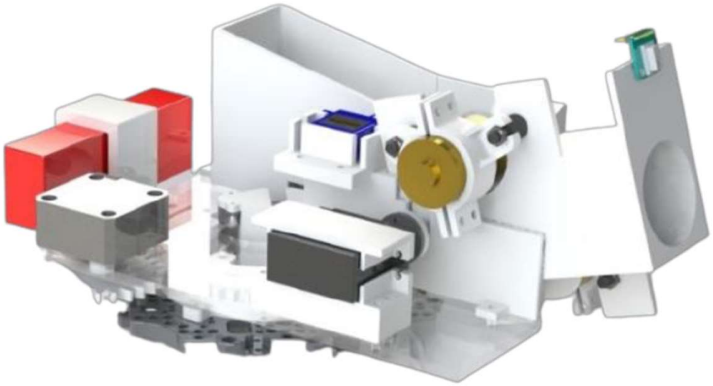
One design used a carousell to feed the balls into a launcher.



Other designs utilized other types of firing and loading mechanisms, namely those that look like a pipe.

Most designs showcased on the website also utilized flywheels as the main source of power for the projectile.

From [alvynben/r2auto_nav](), we observed a single flywheel setup coupled with a servo that manually feeds the flywheel.



From [hjunleon/r2auto_nav]() we observed another single flywheel setup, and another servo that manually fed the flywheel. However, this one also has a stepper for yaw adjustment and a second servo for pitch adjustment.

Sensors and Components

The following components are those we encountered during our research and decided were worthwhile to investigate. They are not an exhaustive list of what we will be utilizing in the final design.

Module	Purpose	Notes
AMG8833	IR Camera, target detection	• Documented here: https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor • Interface: I2C • Voltage: Accepts 3.3V • Resolution: 8x8 matrix • Temperature range: -20°C to 80°C • Frame rate: 10 Hz • Current draw: unknown (most likely negligible as a sensor)
MLX90640 Thermal Camera	IR Camera, target detection	• Documented: here: https://makersportal.com/shop/mlx90640-thermal-camera-for-raspberry-pi-32-x-24-pixels • Interface: I2C • Voltage: 3.3V • Resolution: 32x34 matrix • Frame rate: 32 Hz • Temperature range: -40°C to 300°C • Current draw: 20 mA (negligible)
PN532 (provided)	NFC reader	• Documented here: https://learn.adafruit.com/adafruit-pn532-rfid-nfc • Interface: I2C, SPI • Voltage: Accepts 5V • Current draw: unknown (most likely negligible as a sensor) • Size: 53.3mm x 117.7 mm x 1.1 mm
SG90 (provided)	Servo motor	• Documented here: https://components101.com/motors/servo-motor-basics-pinout-datasheet • Voltage: 5V • Torque: 2.5 kg/cm • Speed: 0.1s / 60°
RS Pro Geared DC Motor,	Flywheel motor	• Documented here: https://sg.rs-online.com/web/p/dc-motors/2389715 • Voltage: 6V-15V

5.75W		• Speed: 10668 RPM • Note: More powerful versions exist, as well as less powerful versions, which may be used if our budget or power requirements restricts us from using this component. However, because the flywheel motor will not be utilized for a large portion of the trial, we suspect power budget will not be an issue.

Concept Design

Base Design

Initially, we had no plans on modifying the base “Turtlebot 3 Burger” model, as we have not found any mounting requirements that would motivate such a change.

We elevated the height of the turtlebot by exchanging the 20mm plate supports for longer ones, creating a space on the second layer of the turtlebot for our ping pong ball launcher to sit in. While making space for the launcher, we shifted the OpenCR and Raspberry Pi board to the third layer. We extended our robot by adding 2 pieces of waffle plates to the front of our robot. The extra space allowed for us to place our NFC sensor in between the 2 waffle pi and space to place our motor and flywheels.

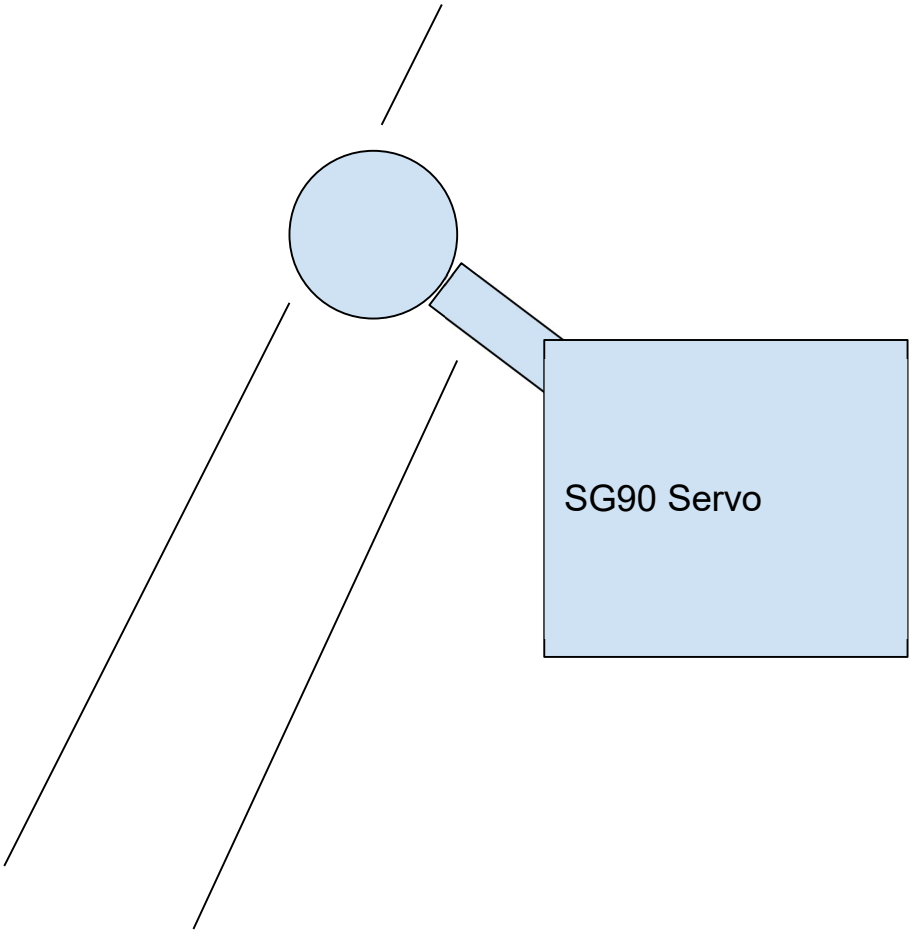
However, this design eventually led to issues with balancing, due to the weight of the motors mounted in the two front plates bringing the center of gravity too far forward.There were four ways we tried to configure the bot — with a front wheel, with just the wheel bracket in front for support, without any front support, and without front support but with counterweight at the back. Using a front wheel caused the wheels to lose traction when turning because one side would be lifted off the ground. Removing the wheel but keeping the bracket allowed the bot to turn but caused issues when running into the NFC tags, which had some thickness that the front support was unable to clear. Removing the support entirely made the bot unstable when coming to a sudden stop. Placing counterweights at the back allowed the bot to stop smoothly but slowed down the bot significantly. During the trial run, we were forced to bite the bullet and install counterweights at the rear.

Subsystem Design

Loading and Storage

Our loading structure uses gravity to feed the balls into the firing tube. The structure is long enough to accommodate 4 ping-pong balls, and is angled such that it does not obstruct the LIDAR. The lower end of the structure is connected to the firing mechanism. A slot in the firing tube was made for a servo arm controlled by the Raspberry Pi to fire the balls. When the turtlebot detects the target and wants to fire, the servo will be spun by 180 degrees then back, pushing the first ball into the flywheels, and stopping the second.

We decided to use this system because it relies the least on moving parts out of all the possible mechanisms researched. By utilizing gravity, the pushing mechanism only has to be located at the end of the hopper. This saves us space and complexity — while the tradeoff in reliability due to the unreliability of gravity, is mostly negligible because the turtlebot is only meant to operate completely upright, meaning we can expect that the balls will be able to feed constantly. This system balances between complexity and maximum capacity, such that the failure rate of loading and feeding is optimized against the failure rate induced by inaccuracy.



Side view

After the balls have been loaded, we expect the TA to press a momentary button mounted on the turtlebot which is connected via GPIO to the Raspberry Pi.

We chose this method because it was simple. Originally, sensors such as a limit switch mounted at the end of the hopper in order to detect when the loading mechanism was full were considered. However, asking the TA to perform a simple action to signal to the turtlebot is much more reliable and easy to implement.

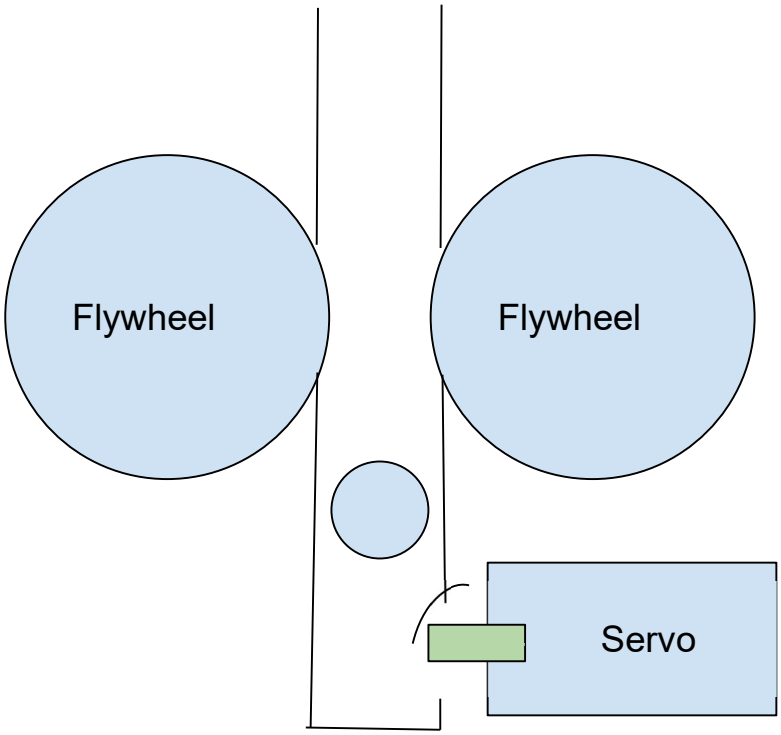
A momentary switch

Firing

We are using a dual flywheel firing mechanism to launch the ping pong ball. They are powered by wiring in series with the 12 volt SMW250-02 socket on the OpenCR board, and a MOSFET transistor, which are controlled with PWM signals from the Raspberry Pi GPIO, coupled with a SG90 servo motor to push the balls. Once the location of the target has been detected, calculations will be done to find the optimal firing position, and the ideal power of the flywheels (controlled by changing the PWM duty cycle). The method of calculating these values was done through experimentation.

A dual flywheel setup was chosen because our power budget allows for them, and weight isn't an important factor. Therefore, we prioritized the precision of the firing mechanism and chose a dual flywheel mechanism.

Midway through the design process, we were informed that the targets would be strictly on the floor only, and that the bot was allowed to move close to the target. In this case, flywheels were unnecessary, utilizing a weaker but easier to construct method of firing, such as a solenoid based firing mechanism, would have been ideal.



Top view

Flywheel Motors

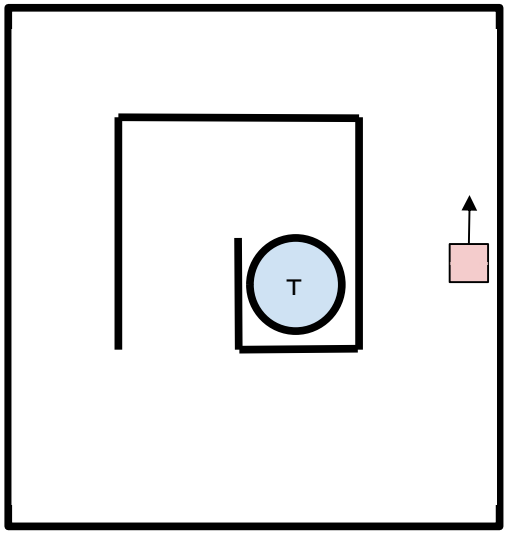
We chose the RS Pro Geared DC Motor which runs at 5.75W from 6-15V. This was chosen because it was the most powerful motor we researched that ran at 12V under the current constraints of the OpenCR board. While other alternatives were considered, they would all have required some form of voltage conversion in order to integrate with the other components, or they drew more current than the OpenCR board could supply.

However, this was not an ideal choice. Because the targets were on the ground, the higher power of these motors when compared to those which ran at lower voltages was unnecessary. This became a problem when considering the surge current of the DC motors. When powering on the motors, they would draw too much power causing the system to brown out. This was fixed by slowly increasing the power fed to the motors by using PWM signals. This worked fine except when the Raspberry Pi was under heavy software load. Because the RPi.GPIO library used software PWM, sometimes the period was abnormally long due to spikes in software latency, which caused the surge current to extend long enough to cause the brown out. Eventually, this was solved by switching to hardware PWM, which was successful in preventing all brown outs from occurring. While there was no impact to the end result by using stronger motors as we kept within budget, using motors with higher power than necessary led to a lot of time being spent debugging this issue.

Navigation

Our navigation algorithm is a wall-following algorithm. This algorithm was chosen due to simplicity, and because the NFC tags were arranged next to the outer walls. This meant that any algorithm used would most likely also have to include a wall following portion in order to guarantee that it finds the tags. In addition, later on we were also guaranteed that the target was adjacent to a wall. Thus, by following the walls, we would also be guaranteed to encounter the target.

Previously, a hybrid wall-following and frontier exploration algorithm was considered. This was because in some types of mazes, following the wall to the right and only turning left when you reach a corner is a guaranteed way to find the exit, eventually. However, in this case, there may be obstacles disconnected from the edges of the maze, which our bot may not explore.

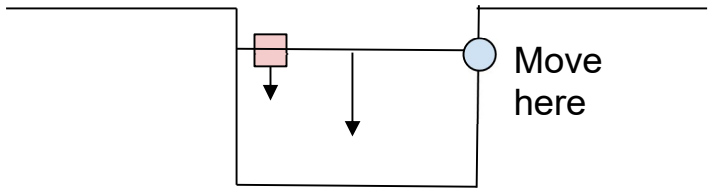


In this example, a robot using a wall-following algorithm will never find the target.

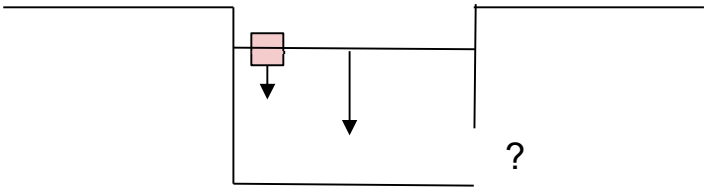
Therefore, we combined the wall-following idea with the concept of exploring unexplored areas of the maze. This is done by recording where the turtlebot has explored using the SLAM data received from the turtlebot. If, when the turtlebot is following the wall, it reaches a position it has been in before, it instead navigates towards an unexplored wall indicated in the occupancy map data.

However, nearing the end of the design process, we were told that this situation was not part of the scope of the task. Therefore, we decided to utilize the simpler pure wall following method.

Midway through the design process, we were also informed that the bot would be evaluated on the timing of map completion. To speed up the mapping, we attempted to program the bot to skip sections that were already completely explored.

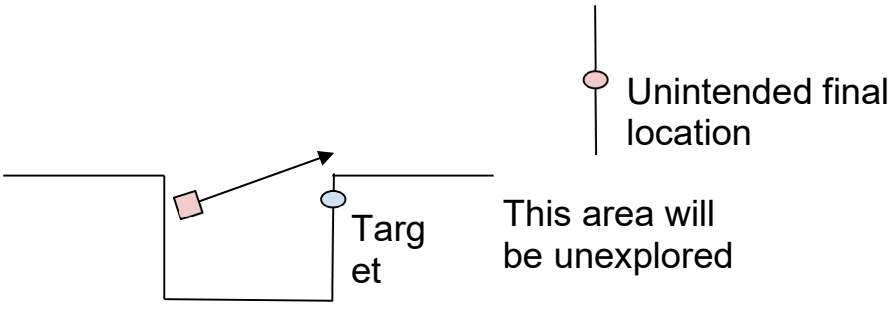


No gaps in maze in front of bot, can skip



Gap is present, cannot skip

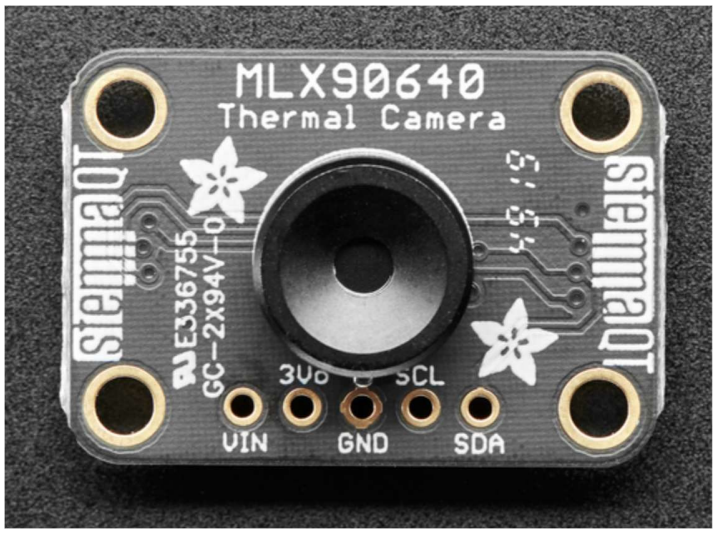
To do this, we draw a line perpendicular to the heading of the bot. Then, we run a flood fill algorithm to determine whether or not the space in front of the bot is completely explored. If it is, then the bot can turn left without skipping the exploration of any walls. This worked relatively well in simulated runs in Gazebo. However, real world tests showed that latency issues caused by network and processing speed limitations would cause the bot to occasionally make too sharp a turn, miss the wall it was supposed to reach, and potentially skip some of the map.



Bot makes too sharp of a turn, and misses the wall it was intended to meet

Due to deadline limitations, the skipping algorithm was not viable and disabled for the final run.

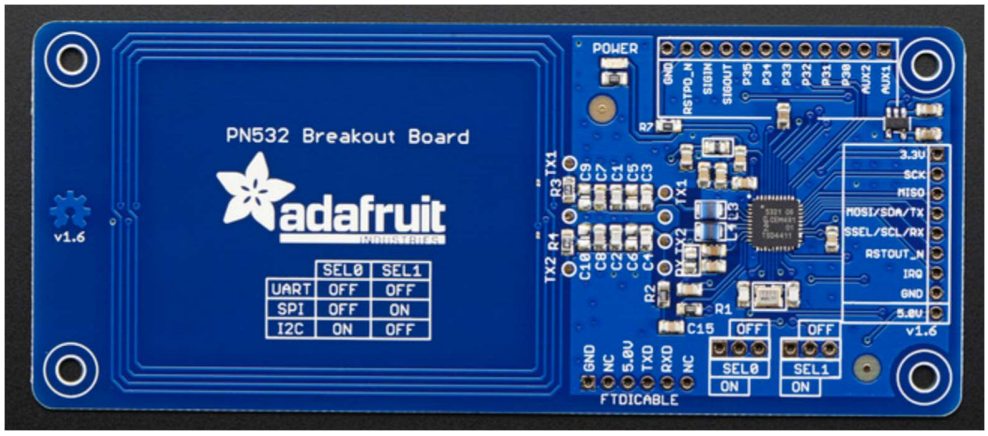
Target Detection and Aiming



Our target detection system utilizes an off-the-shelf component, the MLX90640 IR camera, which allows us to scan the environment for hotspots that indicates the location of a target. The system was calibrated to minimize false positives while minimizing false negatives. This was done by taking multiple samples from different positions once the IR sensor detected the target. In tandem with the LIDAR sensor, the turtlebot uses these samples to pinpoint the location of the target. In order to minimize moving parts, our aiming system was done completely via moving the turtlebot and varying the power of the flywheels.

In deciding to use this component, we compared the MLX90640 IR camera against an AMG8833 IR camera. The AMG8833 was cheaper, but also came with lower frame rate and resolution. The MLX90640 was ultimately chosen because we were able to borrow one from the iDP electronics lab.

NFC Detection



Our NFC reading system utilizes an off-the-shelf component, the PN532, which allows us to scan a NFC tag located on the ground. It will be mounted about 2 cm off the ground. Until an NFC tag is detected, the software constantly scans for a tag. Once a tag is scanned, it triggers the turtlebot to stop and waits for the TA to load. The robot will remain in this mode until a momentary button is pushed, signaling it to resume searching for the target. At this phase, the NFC tag will no longer be used.

This module was chosen because it was provided to us. It was guaranteed by the facilitators of the trial to work with the setup of cards. The other researched components were relatively cheap, but had no clear advantages other than a smaller size. We mounted the module in between the plates which extended from the front of the turtlebot, thus not interfering with the mounting of other components.

BOGAT

Electrical

1. More than 1 sensor might use the same type of serial communication. Need a common circuit board to connect the devices to the SDA and SCL line.

Mechanical:

1. Additional protrusions of the turtlebot will need a larger threshold for wall-following navigation. Protrusions have to be fitted such that the furthest distance from the turning point of the robot (center of both wheels) are approximately equal on all 4 sides of the robot.
2. The circuit board mounted at the top of the board might not have enough space. Shifting of the Lidar forward will be needed to free up space. Algorithms will have to take this shift of lidar into account.
3. Front ball cater without the ball bearing can be caught on items on the ground. Remove it and replace it with counter weights behind to balance the robot.

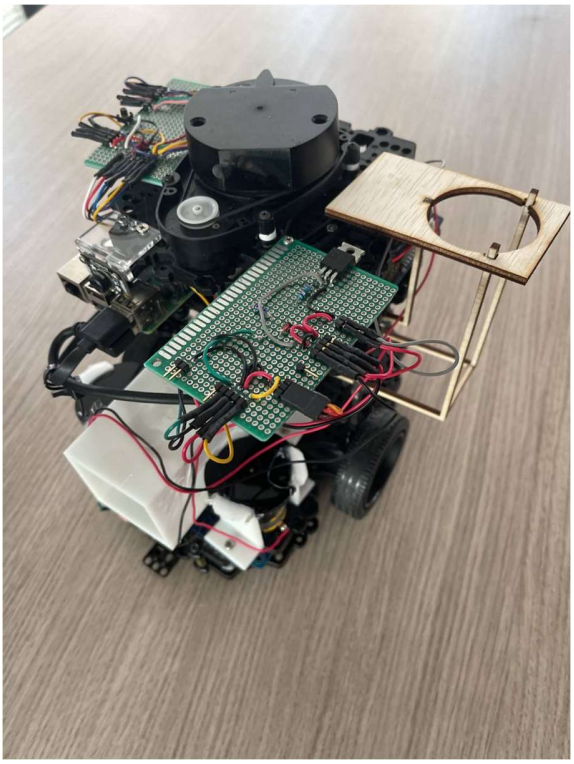
Software

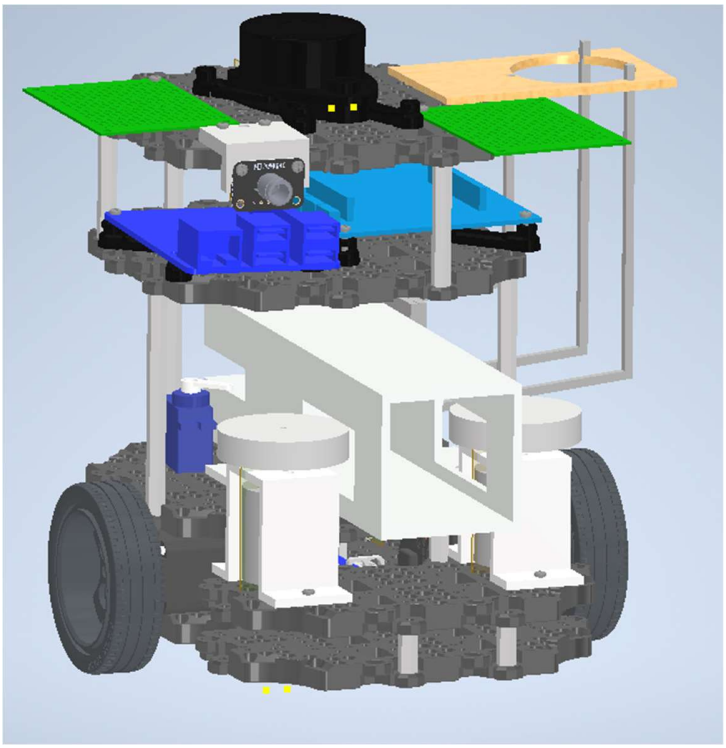
1. Current surge at the start up of the bot might draw too much current from the OpenCR board. Need to implement code to slowly increment motor speed by increasing duty cycle of PWM controlling the motors.

Design

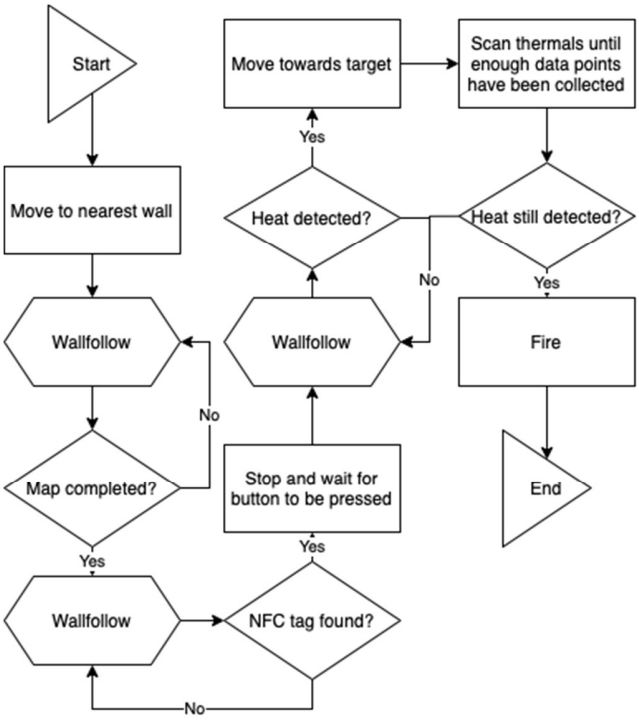
Mechanical Architecture

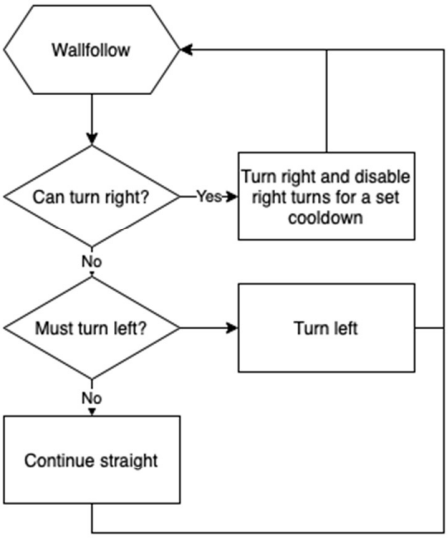
The 3D-printed firing tube was placed between the bottom two waffle plates in order to have a reduced turning radius. We went for a cuboidal structure rather than a cylindrical one because we wanted a flat bottom surface parallel to the waffle plates as it made adjustments easier to incorporate with respect to the gap between the upper and lower waffle plates and the positioning of the two motors. The loading structure, which can hold 4 ping pong balls, is made from scrap wood and the feeding mechanism is based on gravity; the lower part of the loading structure is at a small angle to the waffle plates. Additionally, two half waffle plates were fitted to the front of the turtle bot, one above the other, and includes the NFC mount and the motor setup for firing the balls.



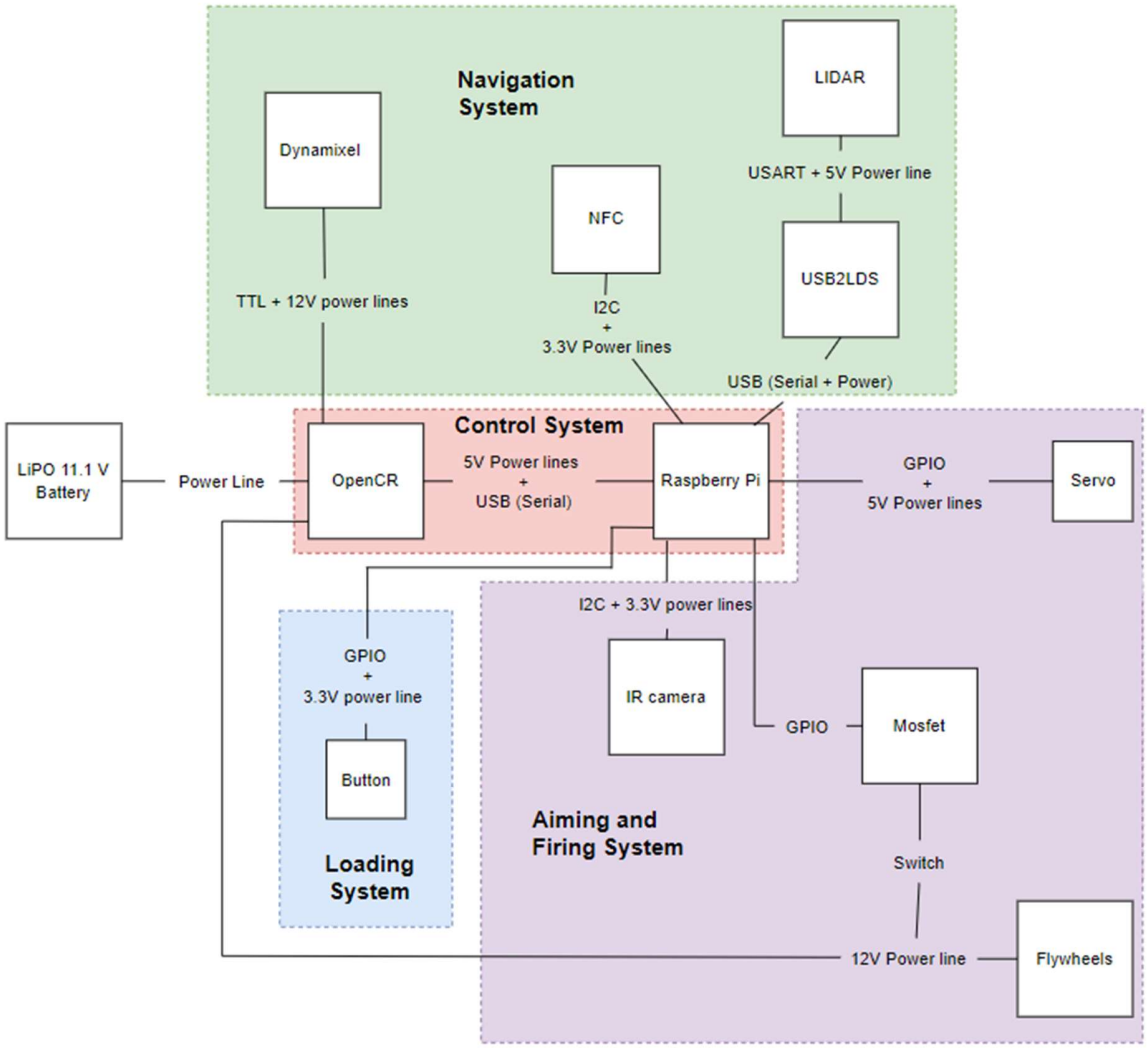


Software Architecture

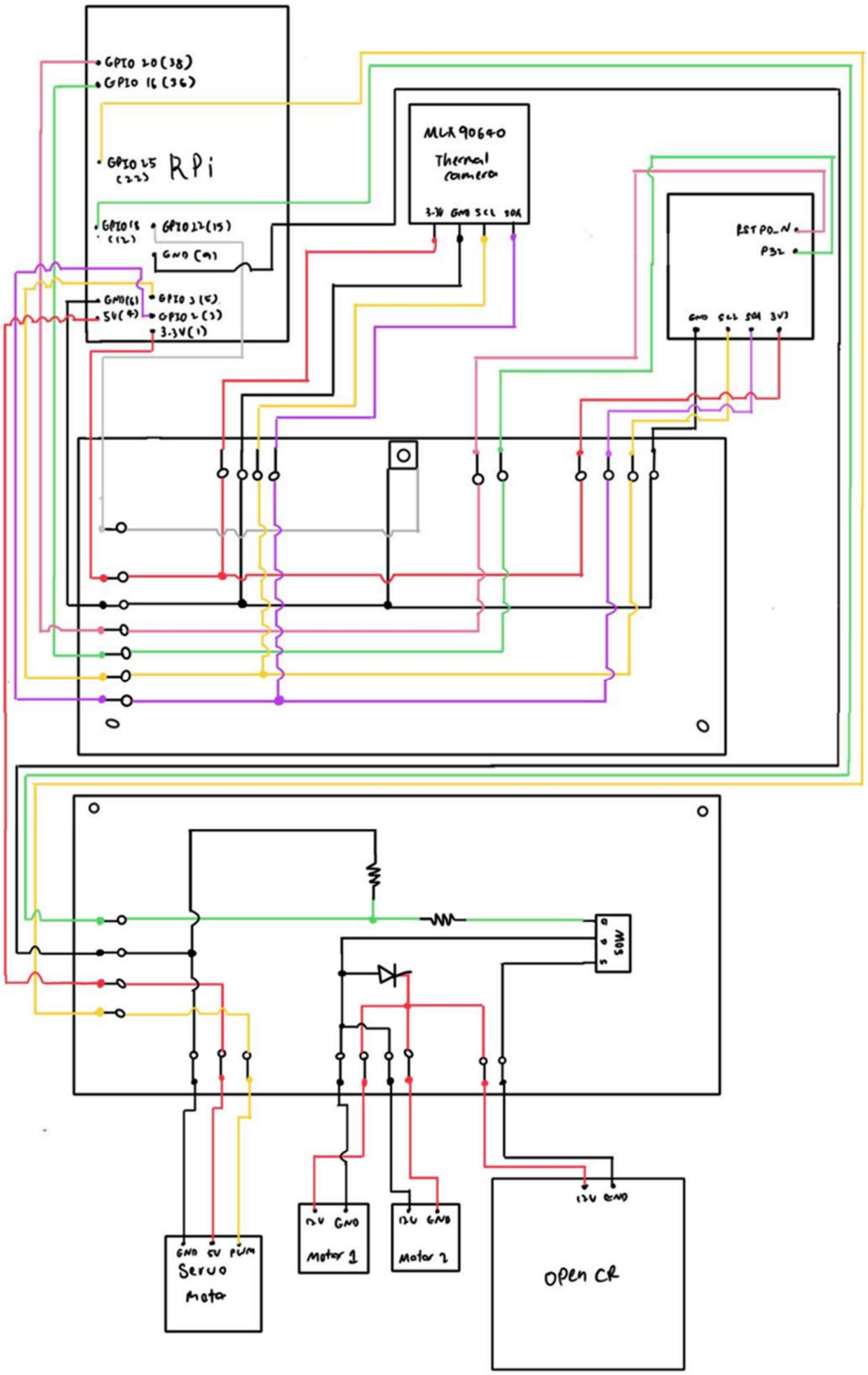




Electrical Architecture



Circuit diagram (exclude peripherals that come with Turtlebot)



Bill of Materials

Item number	Quantity	Description
1	1	Lidar
2	2	Wheel
3	2	Dynamixel
4	1	Li-Po Battery
5	1	Ball caster
6	10	Waffle plate
7	1	Open Cr
8	1	Open Cr cable
9	1	USB2LDS
10	1	USB2LDS cable
11	1	Raspberry pi 3
12	1	Thermal Camera - MLX90640
13	1	Mosfet IRLZ44N
14	2	DC Motor - RS Pro 238-9715
15	1	Servo motor (SG90)
16	1	Firing tube
17	2	Motor bracket
18	1	Thermal camera bracket
19	2	Flywheel
20	1	NFC reader PN532
21	15	Plate supports
22	12	Brackets

23	2	Custom PCB
24	1	Ball Loading Structure

Budget and cost

We were given a budget of $100

	Item	**Quantity**	**Cost**
1.	Thermal Camera - MLX90640	1	Lab (S$0)
2.	DC Motor - RS Pro 238-9715	2	S$30.90
3.	Mosfet IRLZ44N	1	Lab (S$0)
4.	Pipe	1	S$1.40

SETUP

Hardware Setup

Software Setup

Server-side setup (on laptop):

1. Install Ubuntu 20.04 onto the machine that is to be used as a server.
2. (https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/#pc-setup) Follow the directions under the “Foxy” tab to install the “Foxy” version of ros2.
3. Run the following commands.

```
cd ~/colcon_ws/src
git clone https://github.com/EthanYidong/r2auto_nav
sh r2auto_nav/scripts/bootstrap_server.sh
cd ~/colcon_ws
colcon build
source ~/.bashrc
```

This will install the code and libraries necessary to run the code.

Client-side setup (on Raspberry Pi):

1. (https://emanual.robotis.com/docs/en/platform/turtlebot3/sbc_setup/#sbc-setup) Follow the directions under the "Foxy" tab to install the ros2 operating system to an SD card for use on the Raspberry Pi.
2. Enter the authentication details for a wireless network into the cloud-init file as described in the above tutorial. For this step, either a dedicated router or a mobile hotspot may be used, at the user's discretion.
3. Install the SD card into the Raspberry Pi.
4. If the router/mobile phone supports viewing the IP addresses of connected devices, you may skip this step. Otherwise, you must power on the Raspberry Pi while connected to a display. Run the command "`ip addr`" in order to get the IP address of the Raspberry Pi.
5. Power on the Raspberry Pi.
6. On the server laptop, connect to the same network chosen in step 2.
7. Run the following commands.

```
# The password is turtlebot
ssh ubuntu@[ip address from step 4]
cd ~/turtlebot_ws/src
git clone https://github.com/EthanYidong/r2auto_nav
sh r2auto_nav/scripts/bootstrap_client.sh
cd ~/turtlebot_ws
colcon build
source ~/.bashrc
```

8. (https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/#opencr-setup) If the OpenCR board has not been used before, follow these directions to update its firmware.

Testing and Calibration

Each test has a python script that will run the specific hardware associated with it, for the user to verify its operation. Scripts are to be run on the Raspberry Pi.

Component	Testing Procedure
Flywheel Motor	Run `test_scripts/firing.py`. Wait for the flywheels to reach full power. The text "spun up" should be printed to the terminal.

	If the OpenCR board crashes (the boot tone for the board will be played in this case), this means that the surge current of the motors is causing the system to brown out. Ensure that no mechanical obstructions are present by spinning the flywheels with your hands, feeling for any resistance.
Firing Servo Motor	Run `test_scripts/firing.py`. Wait for the flywheels to reach full power. The text “spun up” should be printed to the terminal. Press the enter key on the terminal. The servo arm should extend forward and return to its original position. If the servo motor fails, listen to the motor. If there is no small jittering sound present, that means that the connection is loose. Disconnect and reconnect the servo motor, and try the test again. If there is jittering sound but the motor does not move, ensure that no mechanical obstructions are present.
Full Firing System	Run `test_scripts/firing.py`. Load the bot to full capacity (4 balls). Wait for the flywheels to reach full power. The text “spun up” should be printed to the terminal. Press the enter key on the terminal. A ball should be fired. If the ball fails to fire, remove the servo from the servo mounting bracket. If the balls do not freely roll from the holding bracket to the firing tube, then the bracket must be adjusted. Attempt to realign the holding bracket by heating with a heat gun to reflow the hot glue.
Button	Run using ros2, `ros2 run ts_client button`. Press the button. “Button pressed” should be printed to the console. If not, check the two connections between the button and GPIO 22 and ground on the Raspberry Pi
NFC	Run using ros2, `ros2 run ts_client nfc`. Place the bot over the NFC tag. “NFC detected” should be printed to the console. If an error message is printed, that means that the NFC sensor is not properly connected. Check all the connections.
Thermal	Run `test_scripts/calibrate_thermals.py`. First, place the robot in various locations in which it should not detect a target, and press enter. Next, place the robot in locations in which it should detect a target, and press enter. A reasonable threshold temperature should be output for both. Subtract around 3 degrees from the higher threshold to get the best value to use for target detection.

	If an error message is printed, that means that the thermal sensor is not properly connected. Check all the connections.