Singapore

About Us

We are a team of 3 students from Hwa Chong Institution, Singapore. We have been participating in RoboCup Soccer Open since 2017. We take pride in being the only school in Singapore which does not have a robotics coach, instead relying completely on ourselves to learn, build and program everything from the ground up.

Abstract

Our greatest innovation this year is developing our Raspberry Pi camera system that can track multiple objects at up to 90 FPS, with a client-side GUI that makes it easy for debugging. We have upgraded our robot's processing capabilities by having 3 STM32F1 processors on top of our main Teensy microcontroller, which allows for much more efficient task parallelisation. We have also created our own 2D soccer simulation software with Python for testing out new strategies. Most importantly, we took the time to learn from various teams over the years and have adapted some of their more effective ideas into our robot, such as having a double dribbler design as well as fabricating the mirror from a lathed aluminium tube.

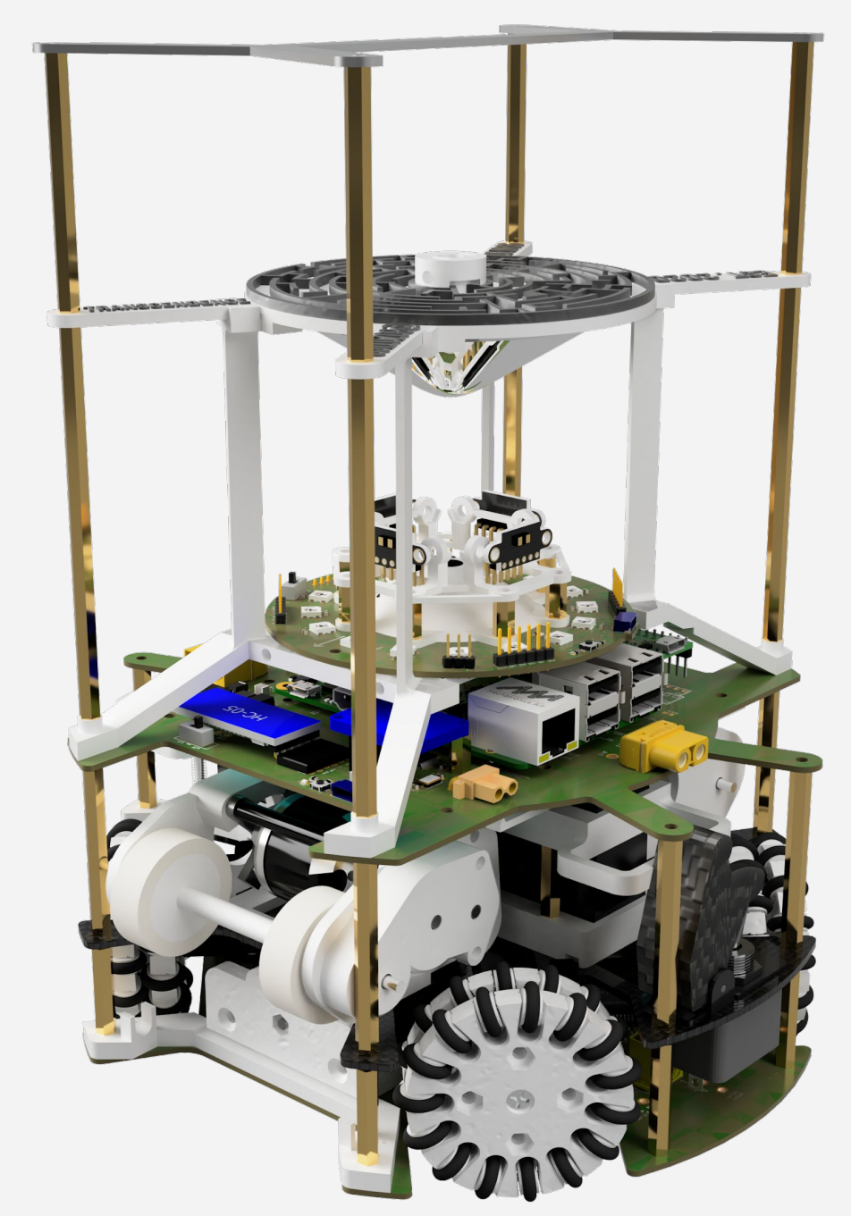
Contact

For more details about our robot, please visit our website at bozo.infocommsociety.com.
Stay updated with our team by following us on Instagram @bozotics and subscribe to our YouTube Channel!

Website **Instagram** **YouTube**

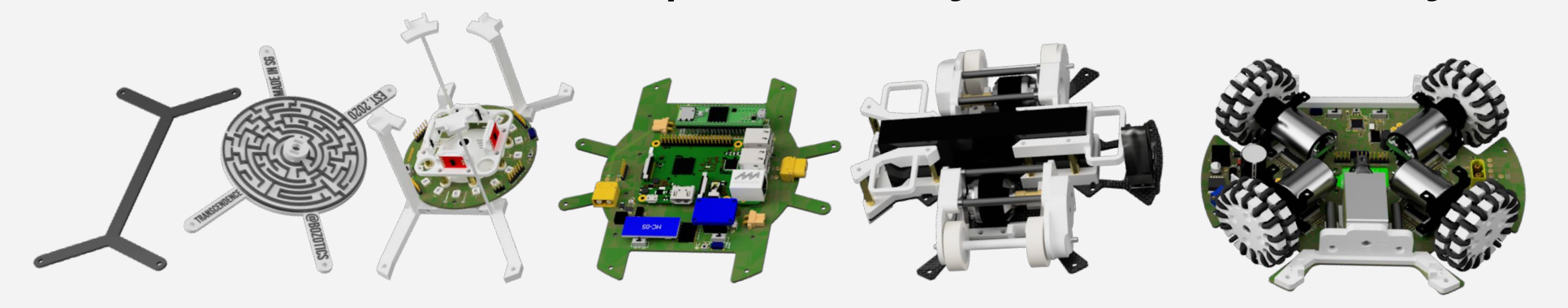
Mechanical

Design Principles

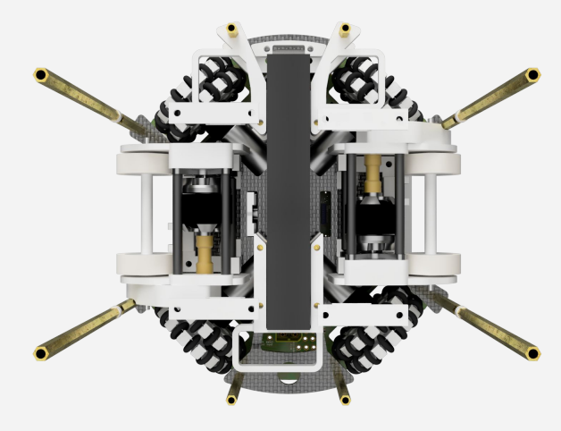


Low Centre of Mass
All the heaviest components are placed on the bottom layers. This reduces skidding issues and allows for more consistent motion.

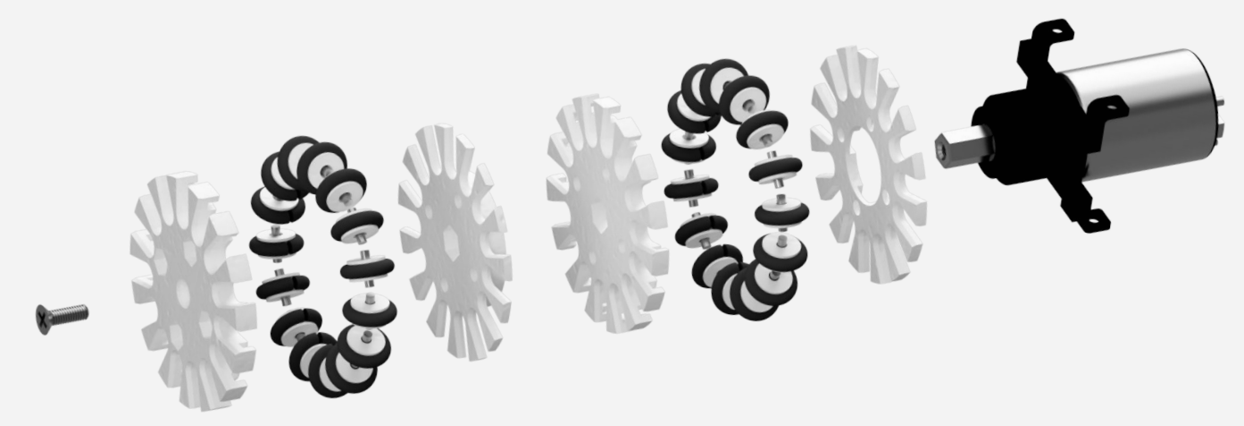
Modular Design
The robot has 6 independently assembled layers.



Even Weight Distribution
The robot is roughly 180° rotationally symmetrical. This ensures equal weight distribution on each wheel, providing for more accurate motion.



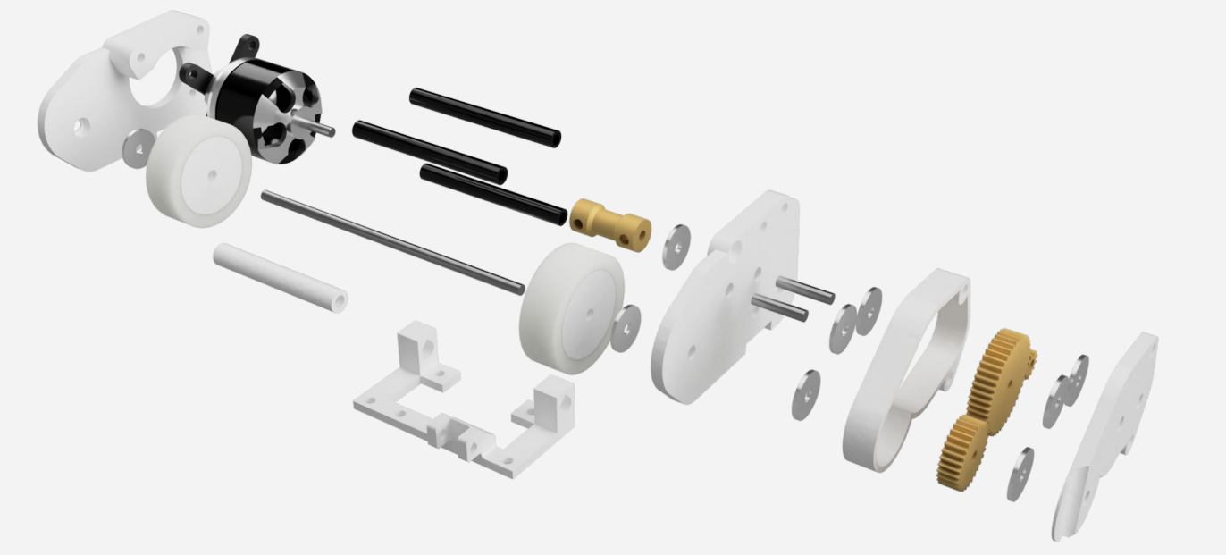
Drive System



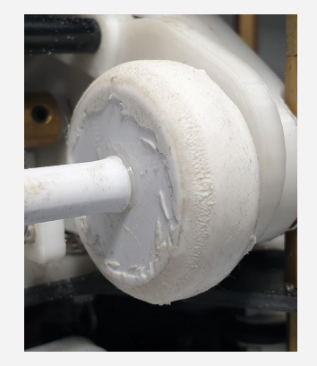
We use 4 JoinMax JMP-BE-3561 motors. It has very high speed (1700 RPM) and torque (1.96 Nm). Our omni-wheels are self-made, mostly out of 3D printed parts.

There are 2 layers of 15 mini-rollers, each fitted tightly with a nitrile rubber O-ring and a short axle. The axles are held in recesses between the 3D printed layers, allowing the rollers to spin freely.

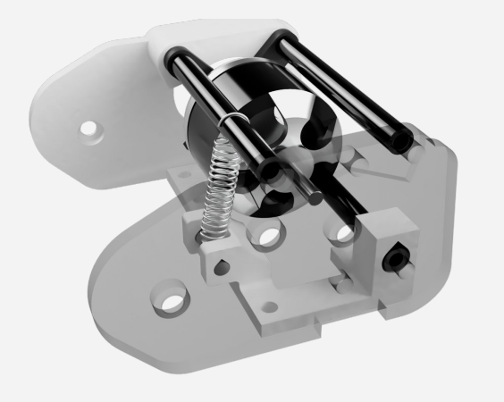
Dribbler



Our dribbler is powered by an 820kV brushless motor with a 2.5:1 gear reduction. It is able to spin the ball at over 2000 RPM.

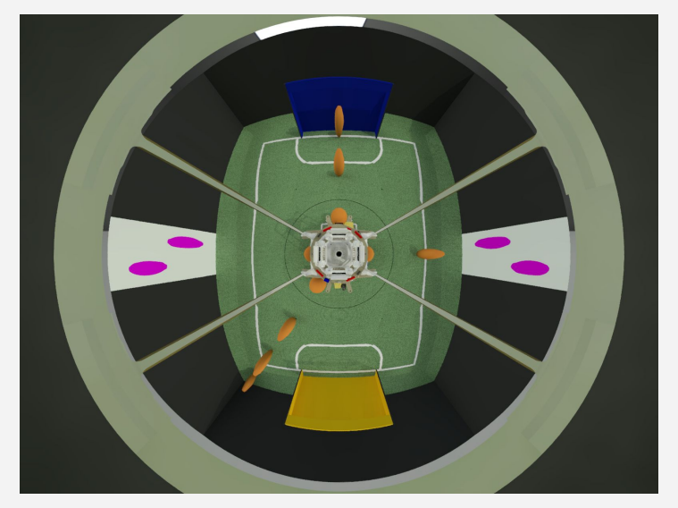


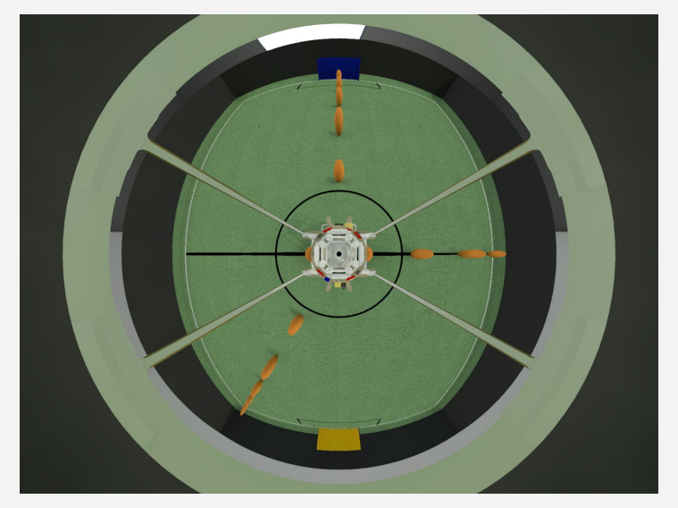
Our rollers are coated with silicone sealant, increasing the friction on the ball.
The dribbler has a suspension system too. It pivots upwards and absorbs the impact of an incoming ball, providing better ball control.



Mirror (Design)

We use a hyperbolic mirror to give the robot a 360° FOV. The profile was designed on CAD and renders were used to tune its shape. Our mirror can see the entire field from any position, even on the SuperTeam Bigfield!


Normal render


Bigfield render

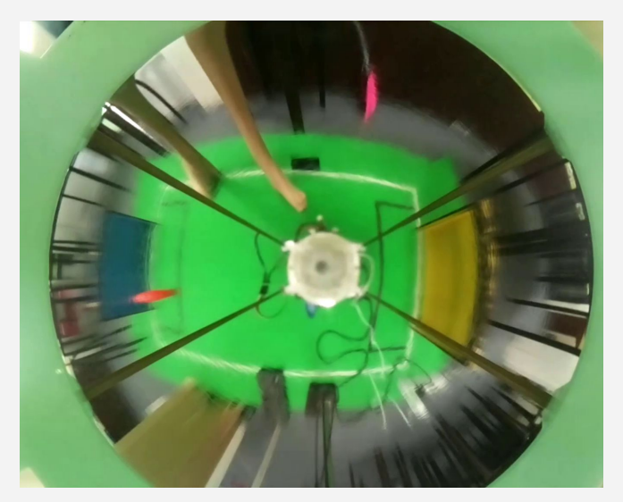
Mirror (Fabrication)

The mirror was machined from an aluminium tube with a CNC lathe, followed by wet sanding and polishing to a mirror finish.

Unpolished

Wet sanding

Final mirror

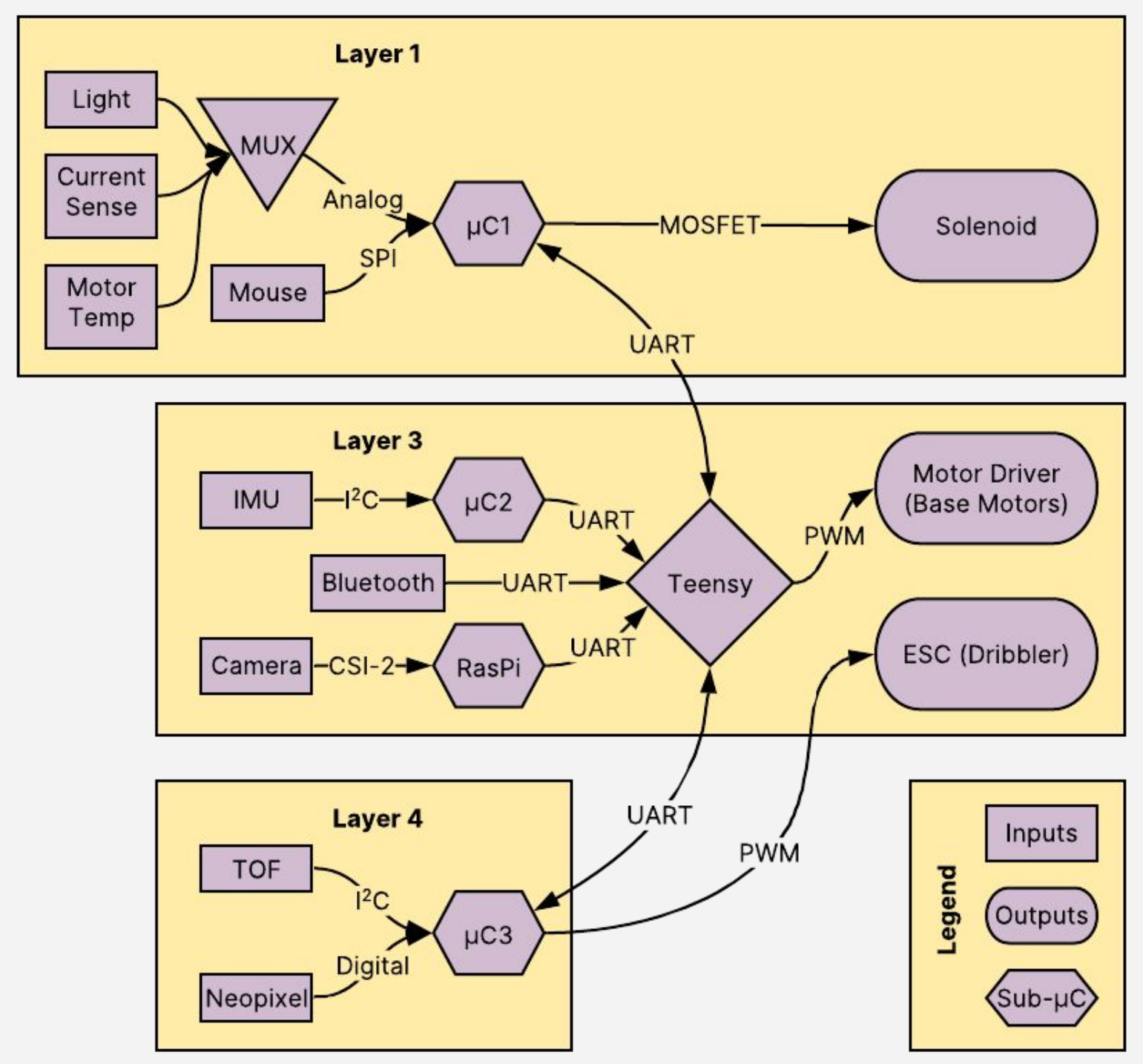

Actual camera view

Electronics

All our electronic components are connected together by PCBs designed in Autodesk EAGLE and fabricated with JLCPCB. To accomodate for the limited board space, the design was made extremely compact by utilising SMD components rather than THT breakout boards as much as possible due to the smaller footprint.

Components List

Component	Cost (SGD)
Layer 1	
JMP-BE-3561 Motor ×4	$480
1040B Solenoid	$5
VNH5019A Motor Driver ×4	$8
ALS-PT19 Phototransistor ×40	$4
PMW3360 Mouse Sensor	$10
Layer 2	
Revolectrix 3S 3300mAh 40C Li-Po Battery	$20
XA2212 820KV BLDC ×2	$20
Layer 3	
Teensy 3.5	$50
STM32F103CBT6 ×3	$6
Adafruit NXP Precision IMU	$25
Raspberry Pi 3B	$45
Raspberry Pi Camera V1	$3
HC-05 Bluetooth Module	$5
Hobbywing ESC ×2	$20
Layer 4	
VL53L1X TOF ×4	$30
WS2812B Neopixel ×16	$2
Others	
Carbon Fiber	$26
PCB (4 plates)	$25
Assorted Parts	≈$60
Total	≈$850

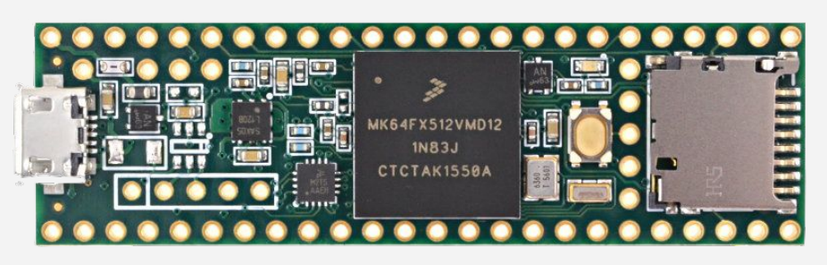


Connection scheme

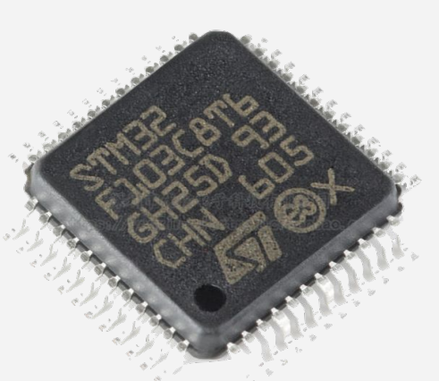
We decided to use UART as the communication protocol between µC as it is the most stable and easiest to implement compared to other protocols like SPI and I^{2}C.

Microcontroller (µC)

We chose a Teensy 3.5/3.6 as our main µC because it has a fast processing speed, small form factor, extensive support on the Arduino IDE, and is easy to reuse as it is on a breakout board.

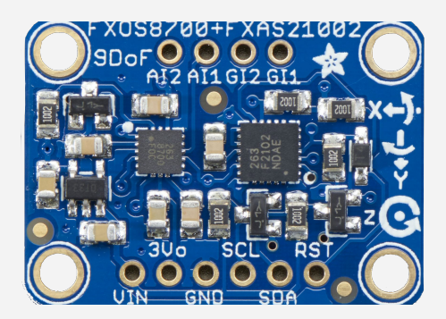


To prevent the Teensy from being bogged down by low-level processes, we use multiple STM32F103 chips, allowing for more tasks to run parallel.



We chose this due to its extensive support in the Arduino ecosystem and low cost.
In order to program these, we use JLink OB modules to flash programs via SWD.

IMU



We use the Adafruit NXP precision IMU. It is connected to an STM32F103 µC which processes data from the magnetometer, accelerometer and gyroscope using NXP's sensor fusion algorithms before sending the robot's bearing over to the Teensy.

Software

Object Detection

To detect objects, we use the Raspberry Pi (RPi) with the RPi camera due to its high processing power and flexibility.
A dynamic priority system controlled by the Teensy allocates the RPi's computational resources efficiently. Commonly used alternatives did not meet our standards: the Pixy had too low resolution, while the OpenMV and Jevois camera has less processing power than our RPi.

Camera

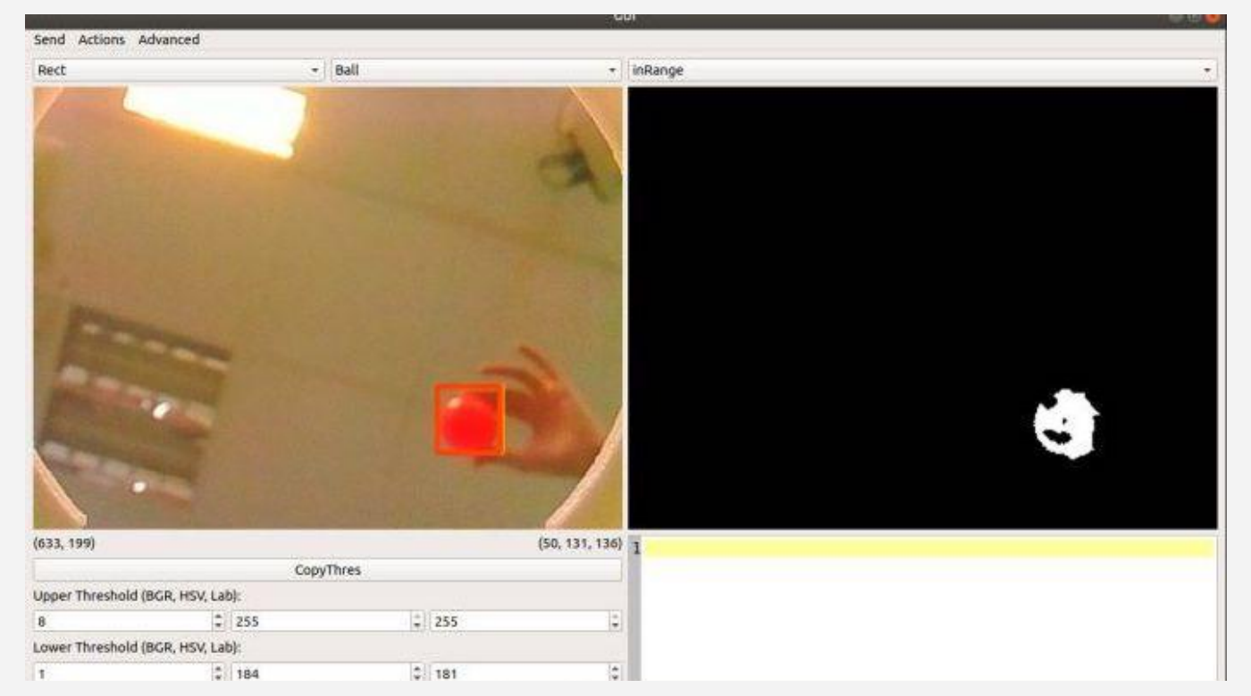
We used RPi camera v1 over the newer v2, since it can work at the full FOV while still achieving a high FPS.
To more accurately locate the ball, we manually tuned saturation and exposure values to better capture the color range of the orange ball - all done via the GUI.

Programming the Raspberry Pi

Most tutorials point towards Python with the official picamera library. However, we code in C++ which compiles into a binary. This overcame latency issues from Python's interpreted nature, while enabling us to bypass the Global Interpreter Lock and utilise true multithreading. We did attempt to use alternative implementations such as PyPy, however we faced many issues and bugs that pushed us towards adopting C++ completely.

Client-side GUI

We wanted to emulate the plug-and-play ability of the Pixy or OpenMV. Hence, we built similar client-side interfaces using Qt and sockets over an ethernet connection. We also integrated SSH so that the process of connecting and transferring of files is automated.



Operating System

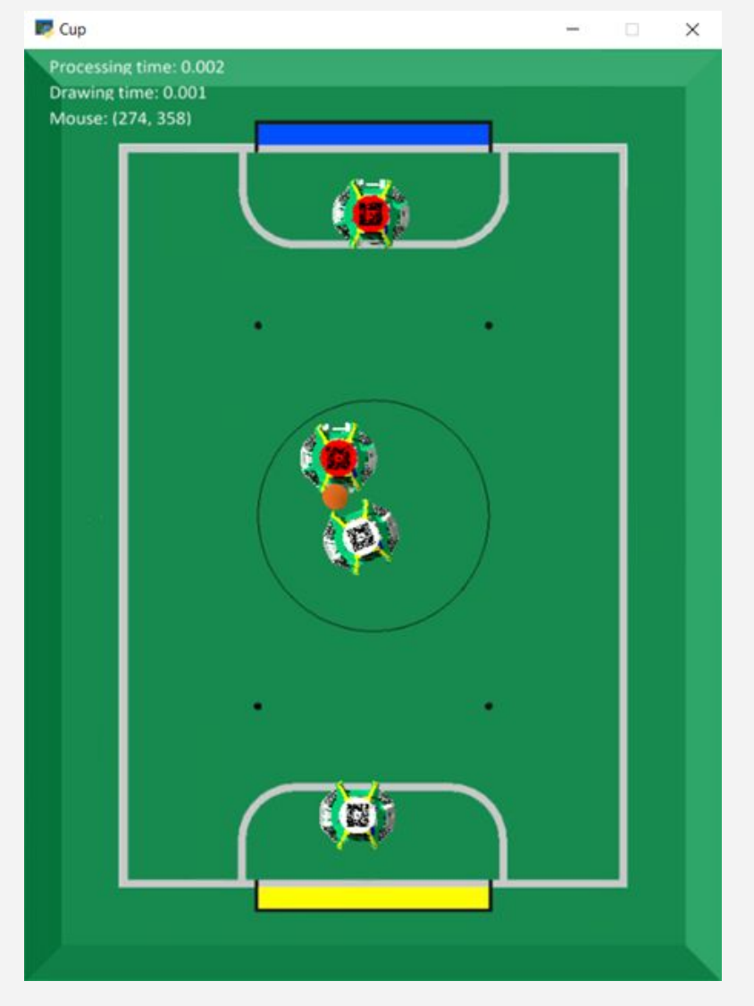
We chose Arch Linux ARM since it has a clean base with not much preinstalled, allowing us to configure the OS with only the programs we require. This drastically decreased boot time (12s) and latency. Looking ahead, tools such as Buildroot can be used to strip the OS down even further and allow us to get a fully embedded system.

Localisation

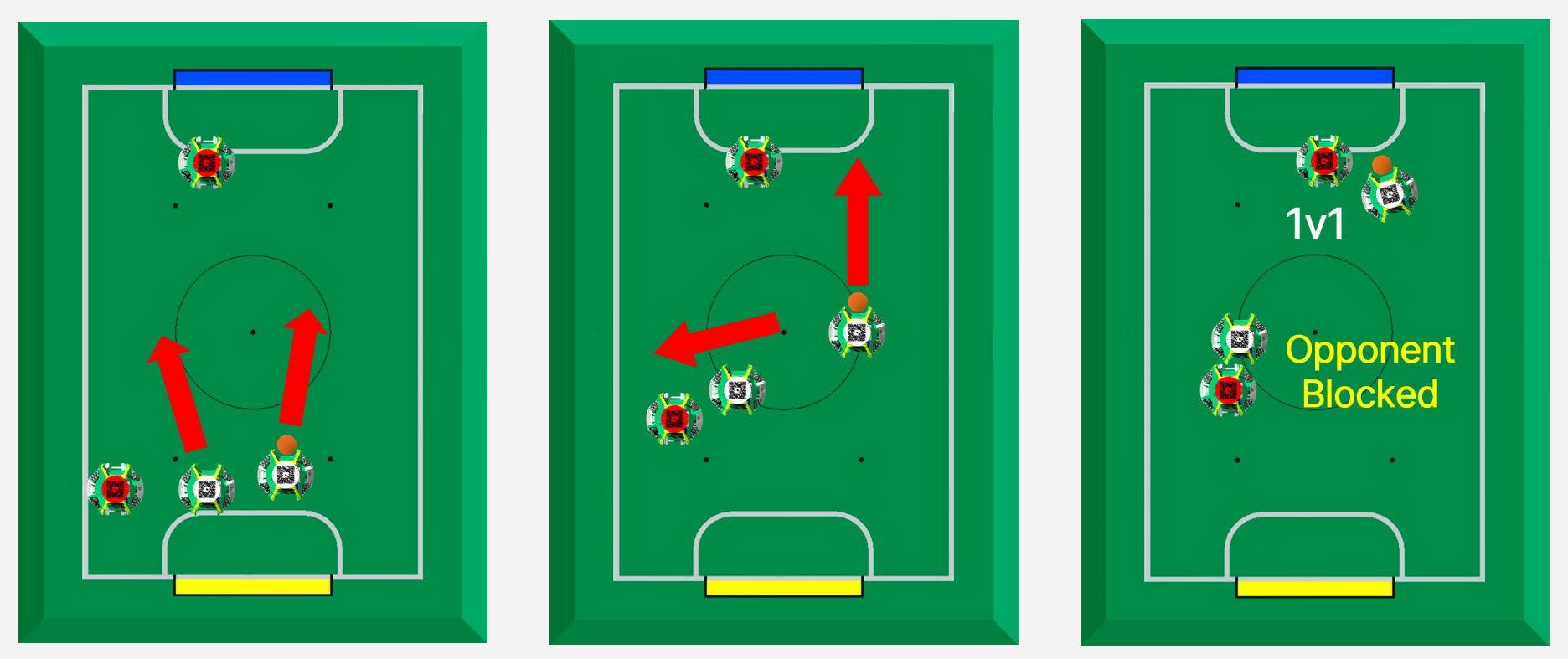
We weight data from the TOF sensors, light sensors, mouse sensor, and camera, based on their computed reliability. This goes through a sensor fusion algorithm to approximate our location on the field, which aids our attack and defense strategy while ensuring the robot stays within the field.

Simulation

We created a 2D simulation with Python to allow us to test our programs without the actual robot. We are also able to test out new strategies that require more complex information such as the opponent's robots' positions. We intend to develop a programmable API for this so other teams can use this software too!


Screenshot of simulation running

A strategy we are experimenting with is "Tackling", where one robot positions itself between the ball and the opponent, thus blocking the opponent's access to the ball and reducing the situation to a 1v1.



Example of "Tackling" situation