League	Soccer Open	Member	Yo Nakano (Kyushu University)	Country	JAPAN	Share
Team Name	Tachyon		Yu Nakano (The University of Electro Communications)			X(Twitter) YouTube Web

This project in being conducted with the support of the Kyushu University fund
Sponsored by : Robert T. Huang Entrepreneurship Center of Kyushu University

Abstract

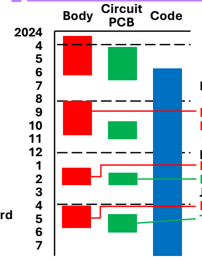
We are a team Tachyon from Japan. Our team is made up of twins who live apart.
Our robots features are unique camera units, new self-localization technology that use image gotten from multiple cameras, and a powerful debugging system that utilizes wireless technology.
We are going to participate this competition with these two nice robots and the strong bond between our team members that will connect us for about 900km.

About us

<Team Members>
Name: Yo Nakano
Role: Body design, Programing
Name: Yu Nakano
Role: Circuit and PCB design, Programing
<Team Motto>
"Let's do anything because this is last game"

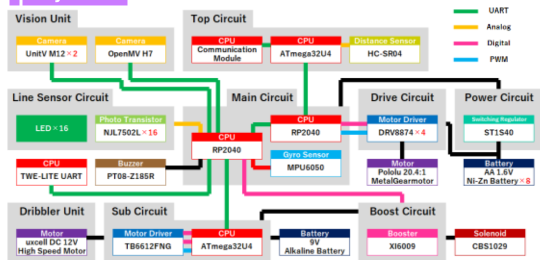
<Record>
Lightweight: Japan Open 2022 - 22nd
Open: Japan Open 2023 - 6th
Open: Japan Open 2025 - 6th
Best Presentation
Special Judge's Award
Sponsor's Award

Time and costs of development



	Cost(Per Robots)
Motor	About $185.0
PCB	About $22.6(Supported by JLCPCB)
Body Frame	About $4.0(Supported by JLCPCB)
Camera	About $159.2
Other Parts	About $70
Total	About $440.8

System



Development tool

Software design
Visual Studio Code(Arduino,C++) Used to create and write programs for RP2040 and ATMega32U4.
OpenMV IDE(MicroPython) Used to create and write OpenMV programs.
MaixPy(MicroPython) Used to create and write UnitV programs.
TWELITE_Stage Used to change the settings of TWE-LITE UART.

Hardware design
Fusion360 Used to create shapes of robot parts and electronic circuit boards.
Kicad Used in the design of electronic circuits.

CNC Machining
For the metal plates of our robot, we designed 3D data using CAD and then outsourced the CNC machining to an overseas company.
PCB Manufacturing
Regarding the printed circuit boards (PCBs), we created the PCB design files, including schematics and wiring diagrams, and then commissioned an overseas company for both PCB manufacturing and surface mounting of ultra-small components.
3D Printing
We produced the robot's intricate parts and fixtures using 3D printers. For this, we utilized the 3D printers owned by our team.

Remote development

The 900-kilometer challenge
Our team are comprised of members who all live in Japan, but their houses are separated by a distance of about 900km. It may be hard to imagine, but for example, Paris and Berlin are about 900km apart. The problems that arise when developing a robot remotely are listed below:

1. Management and Sharing of Robot-Related Data
When developing a robot as a team, there are frequent needs to share robot 3D data and program source code. However, because of the distance separates us, we cannot directly convey information or share files via USB memory sticks.
2. Schedule and Fund Management
For example, when a competition is a few days away, team members would typically confirm the development workflow leading up to it. Also, managing development costs is smoother if direct money receiving is possible. However, if team members live in places where they cannot easily meet, doing it is impossible.
3. Trust in Teammates
In team development, mutual motivation often stems from seeing teammates work. This means there are frequently risks that conflicts between teammates may occur if teammates are in places where they can't see each other, for example, they can only assume that each other is working hard for the team from afar.
Why remote development was successful?
Our remote development was achieved through various ingenious methods. For sharing robot data, we utilized cloud-based data management tools such as Fusion 360 and GitHub. Furthermore, for managing schedules, funds, and others, we employed a service called Notion. By consolidating all our project's information within this service, we were able to check our schedule and grasp issues effectively. In addition, for our project, we selected a project manager from among our members and established a system where the project manager oversees the team. Regular meetings also allowed us to keep abreast of each other's progress and divide work roles, leading to efficient robot development.

Fusion360 GitHub Notion

▲Managing schedules with Notion

<Camera> M5Stack UnitV M12
We use these cameras to let robots know situations of a soccer field. One of the merits of using these cameras is that it can process image using two cores.

<Camera> OpenMV H7
We use a camera to let robots know situations of a soccer field. This camera has high processing power and has ever been used by many teams.

<motor> Pololu 20.4:1 MetalGearmotor
These motors are used to move the robot. We could place motors on a robot at 90 degree increments and wheels stay within the size limit.

<motor> Uxcell DC 12V High Speed Motor
This motor is inexpensive, yet powerful and compact enough to be used as a dribbler's motor.

<Solenoid> TAKAHA CBS1029
The robot kicks the ball with the device. This solenoid is very compact and easy to install on a robot.

<Body Frame> A7075
A7075 is a type of duralumin that is a lightweight, high-strength alloy, making it an excellent material for robot frames. The surface of the frames is anodized to prevent metallic shine.

<Battery> Ni-Zn Battery AAA1.6V 2600mWh
These batteries have a higher voltage than other types of AAA batteries. (Alkaline: 1.5V, Ni-H: 1.2V)
Also, they are said to be safer than Li-ion batteries. The number of our team members is small. So, we chose these safe batteries.

<Battery> 9V Alkaline Battery
This battery is used to power the kicker and dribbler. It is compact and can carry high voltage electricity.

<CPU> ATmega32U4
We use these CPUs to control the dribbler and to get values from the distance sensor. These have the advantage that they can be wired directly to the USB port when mounted on the board.

<CPU> RP2040
We use these CPUs to determine the direction of the robots moving in which motors will rotate based on the values get from each sensors. These have dual cores and high-speed programs.

<Gyro Sensor> MPU6050
We use this sensor to determine the direction the robot facing. We have integrated the circuit into the main board based on the open source of MPU6050 circuit diagram.

<CPU> TWE-LITE UART Blue
We use this CPU for wireless communication between robots and the debugging software "WDS". This device is easy to use because it uses a very simple communication method in which UART communication messages received from the RX pin are output from the TX pin of another device.

<Photo Transistor> NJL7502L
The robots detect the lines by using the change in resistance of the device as it receives different amounts of light reflected from the floor.

<Motor Driver> DRV8874
This is a motor driver that controls a motor based on one digital input for direction command and one PWM input for speed command. There are many chips similar to this one, but its feature is its compatibility with high currents.

<Motor Driver> TB6612FNG
This is a motor driver that controls a motor with two digital inputs and one PWM input. This single chip can control two motors. The robot currently has only one dribbler installed, but it is designed with future expansion in mind.

<Distance Sensor> HC-SR04
This sensor emits ultrasonic waves and measures the distance to an object by calculating the time it takes for the waves to return to the robot. We're using it to determine the robot's position on the field.

<Switching Regulator> ST1S40
This switching regulator integrates peripheral circuits such as switching elements and MOSFETs. By connecting this chip with appropriate external resistors, capacitors, and inductors, you can build your own switching regulator circuit.

Stabilizing movement

We have added some interesting features to the program to let the robot's access to the ball smoother. When the distance between the ball in front of the robot was quite large, the following issues occurred:
1-The camera detects the ball.
2-The robot moves forward towards the ball.
3-The camera loses sight of the ball.
4-The robot steps back to return to own goal.
This phenomenon prevented the robot from smoothly accessing the ball, so as a solution, when the ball was lost, the data was filled by with the most recently detected ball position information. The robot now quickly reaches balls even from a long distance.
However, if the robot considers the ball visible for too long, it will not be able to return to its own goal, so the time limit is set to 1s.

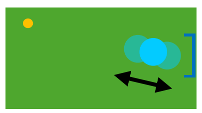
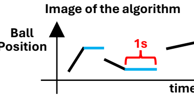


Image of the algorithm

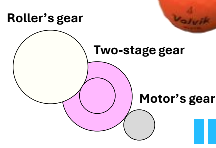


Dribbler & Kicker Unit

Dribbler Unit
Our robots can carry the ball with a dribbler on its back. This unit consists of a motor, a roller, two shafts, and gears, and the gears are used to transmit rotation from the motor to the roller. By using two-stage gears, a high reduction ratio can be achieved in a narrow range. The rollers are fitted with silicone tubes to increase friction with the ball.
Kicker Unit
When our robot confirms through the camera that the ball has landed in the front pocket, it sends about 35V of electricity from the boost circuit to the solenoid, kicking the ball and aiming for a quick shot to a goal.
Removing the unit
"Since both units are secured by only two spacers, the units can be easily removed by taking out the screws that hold the spacers in place."


▲Dribbler


▲Kicker



▲Installation diagram of gears

Searching for balls with a camera

Ball detection is performed by searching for the color of the ball within the scene captured by the camera. The robots extract the ball's color components from a captured image and search for objects with colors similar to that reference color within the captured scene. In ball detection, the three cameras operate independently, with each camera identifying the area within its view where the reference color appears with the largest area as the ball. It then calculates the size of this area and its relative angle to the robot, transmitting this information via serial communication to the robot's control microcontroller.

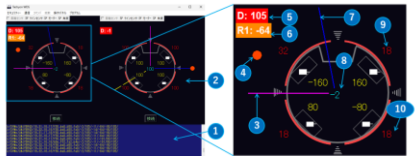
Consideration of ball position estimation algorithms
Previous System (Method 1)
In Method 1, the system would check each camera one by one for the presence or absence of a ball. If a ball's color was recognized by any camera along the way, the robot would perceive the ball to be in that direction and move accordingly. However, this program had a problem: if a camera that was earlier in the color detection sequence reacted to a slight color nuance and cameras that should have detected the ball were ignored, leading the robot to move in the wrong direction.

▲Method 1

System for Comparing Three Visual Information Sources (Method 2)
Therefore, we developed a detection program aimed at solving the aforementioned problem (Method 2). Unlike the previous program that checked camera information sequentially, this program continuously acquires information from all three cameras. It then trusts the information from the camera that sees the ball most clearly, recognizing the ball to be in that direction. This prevents the system to misidentify the ball detections even if a camera reacts to a slight color, because the actual ball will appear larger.

Wireless debugging

WirelessDebuggingSystem
~WDS~
"WDS (Wireless Debugging System) is a robot wireless debugging system developed by us, 'Tachyon.' This system is primarily composed of devices like the one shown in the figure on the right. WDS can be used by employing a dedicated transceiver and dedicated software with the robot."



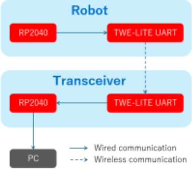
▲Displaying information from robot sensors

1 Serial monitor 2 Sensor panel 3 Robot direction
4 Direction in which the ball was found 5 Distance to the ball
6 Direction in which the ball was found
7 Direction of the blue goal 8 Angle of the robot
9 Line sensor readings 10 Motor power

Development of Windows software
The dedicated Windows software primarily allows for the following actions:
1. Displaying robot sensor information
2. Sending commands to the robot.
We also developed of this software, using the Python programming language

Python

Acquisition of Robot Sensor Readings
WDS requests the robot to send sensor readings once at a fixed interval. The data, sent in a dedicated format, is processed by the software and reflected on the screen. It is possible to acquire sensor readings in real-time, with almost no delay.




▲Transceiver

Windows Software Source Code Release
We have released the source code for the software we developed on GitHub to allow many people to learn about and utilize its command parsing algorithm and screen display technology.
We hope it will be useful for developing new wireless debugging software and other software.

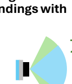
Problems with Conventional Robot Debugging Methods and Their Solutions
Many teams debug robots via wired communication. However, acquiring information from a robot in motion is impossible with wired communication. By contrast, using wireless communication allows for obtaining information from a robot actively tracking a ball, as the communication cable no longer poses an obstacle. With the introduction of WDS, we are now able to quickly identify the causes of robot malfunctions and resolve them.

Vision Unit

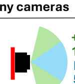
Recent Vision Units
Currently, robot vision units are changing from those using a camera and an omnidirectional mirror to those that do not use omnidirectional mirrors and use cameras to see the surroundings directly. We listed the advantages and disadvantages of looking directly at the surroundings with cameras.

Advantage
-No need to make an omnidirectional mirror
-Objects look more bright and more large
Disadvantage
-Need to use many cameras

Problem
Many teams use four or more cameras. It is because commonly used cameras have HFOV(Horizontal Field of View) below 90 degrees. For example, when we measured the HFOV of the cameras we use, the OpenMV H7's one was about 57 degrees, and the UnitV M12's one was about 82 degrees.



▲UnitV M12 HFOV

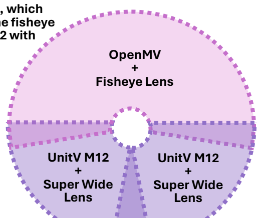


▲OpenMV H7's HFOV

Improvement
Previously, a team had developed a method to view the surroundings by attaching a fisheye lens to the front of the lens of the Pixy camera vertically downward instead of an omnidirectional mirror. We were inspired by this and thought that if we used a fisheye lens or wide-angle lens at the front of the camera lens, we could expand the field of view and see in all directions with fewer cameras. We used a fisheye lens and Super Wide lenses for smartphone cameras, which are easily available online. Combining the OpenMV H7 with the fisheye lens have HFOV of 153 degrees. Also, combining the UnitV M12 with the Super Wide Lenses have HFOV of 128 degrees.

<Lens> hvYOUR Own Super Wide 0.4X
<Lens> hvYOUR Own Fisheye 235

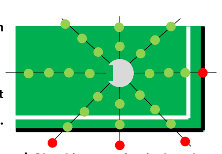
For this reason, we developed a vision unit that can view everything in all directions using three cameras, as shown in the figure on the right.

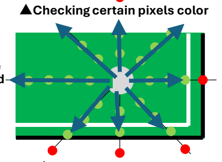


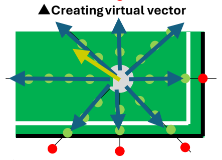

▲OpenMV H7 + Fisheye Lens ▲UnitV M12 + Super Wide Lens

Self-location estimation

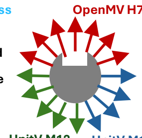
Our robot estimates its position by reading how much green is spreading in 16 directions from the robot.
Detecting field spread
In each direction there are color check points every 100mm up to 1000mm from the robot. The robot checks the color of the pixel in the camera image that corresponds to this point. If it meets the field color threshold it knows that the field extends to that point. The camera does all of this and sends the distance the field is spreading in each direction to the main board, and the rest of the processing is done by the RP2040.
Creating Vectors
The robot then creates a virtual vector for each direction, the magnitude of which corresponds to the distance the field extends in. So in the direction the field extends further it creates a larger virtual vector.
Synthesizing vector
Finally, the robot synthesizes the vectors that represent each direction. The resulting vector will point approximately toward the center of the field. The robot uses the magnitude and direction of this resultant vector to calculate its position on the field or the direction to avoid it if it touches a line.


▲Checking certain pixels color

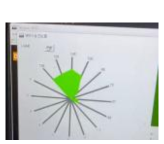

▲Creating virtual vector

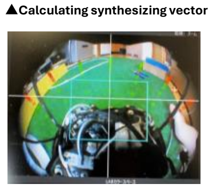

▲Calculating synthesizing vector

Camera sharing the process
This processing is shared among three cameras. As mentioned above, the field spread is checked using the three cameras, and the data is integrated by the RP2040.



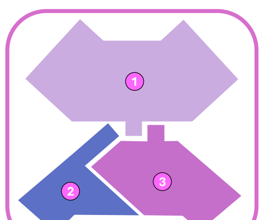
Calibration using WDS
With WDS, you can check the extent of the field seen by the robot on one screen, making it easy to see it, for example, the color threshold of the field of the front camera is incorrect



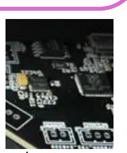

▲Checking certain point's color on OpenMV IDE

Circuit & PCB

PCB Modularization
The circuit boards installed in the robot that serve especially important functions—the 'main circuit' and the 'drive circuit'—and that delivers electricity to various parts of the robot—the 'Power circuit'—are independently structured, as shown in the diagram on the right, a puzzle-like configuration. This design allows each board to be individually changed in case of a failure (e.g., burned IC),



1 Creating a custom circuit of the MPU6050 and Raspberry Pi Pico
Using the schematic of the Raspberry Pi Pico as a reference, We made a custom circuit onto a PCB. Additionally, in a competition two years ago, there was an incident that the robot lost its direction due to a suspected connection failure in the MPU6050 module. Building a custom MPU6050 circuit on the PCB based on a MPU6050 module schematic, we were able to eliminate the issue caused by poor connectivity.


▲main board

2 Creating a custom switching regulator
Because commercially available switching regulator modules were difficult to fit into the robot's limited internal space, we created a custom design
switching regulator using the power IC 'ST1S40.'
Since this IC already integrates components such as MOSFETs, it allowed us to mount a PCB even within the narrow space.

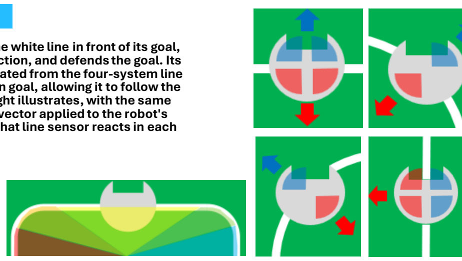
3 Creating a custom driving circuit
One of the challenges was how to position the circuit board within the robot's narrow internal space. Commercially available motor driver modules were too tall to be mounted. Then, we created a custom motor driver board using motor driver ICs. This allowed the PCB to be shaped to fit the available space inside the robot, and we could install the circuit.


▲driving circuit

Defense Program

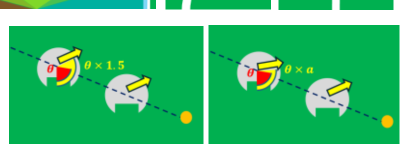
The defensive robot follows the white line in front of its goal, moves towards the ball's direction, and defends the goal. Its approximate position is estimated from the four-system line sensor and the angle to its own goal, allowing it to follow the white line. The figure on the right illustrates, with the same color, the line sensor and the vector applied to the robot's direction of movement when that line sensor reacts in each scenario.

The area in front of the goal is divided into five sections. Based on a rough estimation of the robot's own location, the robot determines its defensive position.



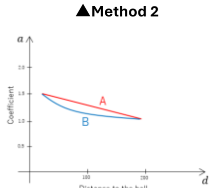
Attacker Program

Initially, we considered moving the robot in a direction 1.5 times the relative angle between the robot and the ball (Method 1). This approach allows for accurate tracking when the ball is in front, and also enables the robot to move behind the ball when it is located to the rear. However, a problem with Method 1 is that it does not incorporate the distance to the ball in determining the direction of travel. Consequently, even if the ball is spotted far away, the robot will move a long distance while still maintaining that long distance to the ball.



▲Method 1 ▲Method 2

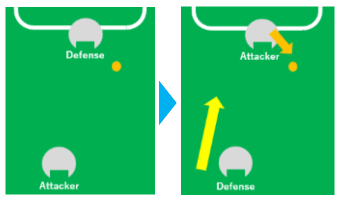
Therefore, we are now considering Method 2. This method involves calculating a coefficient to multiply the angle by, based on both the ball's angle and distance. In this calculation, we are contemplating whether this coefficient should be directly proportional(Graph A), inversely proportional(Graph B), or follow a function of distance. We intend to conduct further validation to find the answer to this problem.



Switching between attack and defense

The robots are assigned roles: an attacker that aggressively tracks the ball and a defender that guards the goal. Furthermore, these roles are adapted flexibly according to the situation. If the ball is near the defensive robot and far from the attacking robot, the robots exchange roles with each other. This leads to quicker attacks and increases the chances of scoring.

During a robot role exchange, the robots communicate wirelessly with each other, and their roles are changed so that they each take on a different responsibility. Furthermore, if one of the robots receives a penalty, the system detects this and the remaining robot will guard the goal, overcoming the disadvantageous situation.



dribbler technique

Ball Capture with the Dribbler
When an attacker robot determines that directly accessing the ball is significantly faster than moving behind it during ball searching, the robot captures the ball with its dribbler. This determination uses information about the ball's relative angle to the robot and the distance to the ball.

Shooting Using the Dribbler
Once the ball is captured with the dribbler, the robot moves the ball towards the center of the open area of the goal. Then, when the distance to the goal falls below a certain threshold, the robot quickly rotates its body 180 degrees, releasing the ball from the robot, and uses the remaining spin on the ball to score a goal.