RoboDragons 2026 Extended Team Description

Masahide Ito, Kentaro Tachi, Kota Sugiura,
Yasuyuki Suzuki, and Yoichi Yamazaki

School of Information Science and Technology, Aichi Prefectural University
1522-3 Ibaragabasama, Nagakute, Aichi 480-1198, JAPAN
Email: ssl.robodragons@gmail.com
Website: https://robodragons.github.io/

Abstract. *RoboDragons* are a team of the RoboCupSoccer Small Size League from Aichi Prefectural University, Japan. This paper shares two technical topics with respect to our system updates between 2025 and 2026: one is to improve team member on-boarding by using Docker tools and another is to present an image-based visual servoing controller via local vision system. The former update was demonstrated for new comers; the latter update was evaluated by an experimental result using an actual RoboDragons robot of the seventh generation.

1 Introduction

RoboDragons are a team of Aichi Prefectural University (APU) participating in the Small Size League (SSL) of RoboCupSoccer. This team originated from *Owaribito*—a joint team between APU and Chubu University—which was founded in 1997. In 2002, since two universities have been ready to manage each individual team, APU built a new team, RoboDragons. After that, RoboDragons have been participating in the SSL for almost quarter of a century including activities as *CMRoboDragons*—a joint team with Carnegie Mellon University in 2004 and 2005. Our best record was the second place in 2009. We also finished thrice in the third place (2007, 2014, and 2022) and five times in the fourth place (2004, 2005, 2013, 2016, and 2025). The number of human and robot members working in team RoboDragons 2025–2026 is over 50 and over 20, respectively (Fig. 1). We have been currently maintaining two generations—the seventh- and eignth-generation robots—since 2024.

This paper provides two technical updates developed between 2025 and 2026. Section 2 describes the improvement of team member onboarding by exploiting Docker tools; Section 3 provides a kinematic model of SSL robot with local vision and a controller for image-based visual servoing.

2 Team Member Onboarding by Using Docker Tools

2.1 Backgroud and Motivation

In radically development approaches like agile development, development environments optimized for individual developers based on their preferences and

Fig. 1: All members in RoboDragons (taken at our home field on 11 June 2025).

experience tend to be adopted. In the development of the SSL field, in particular, goal setting itself tends to become exploratory in response to changes in the competition environment and technical requirements, demanding continuous and fluid improvement. Therefore, it is thought that many SSL teams build their systems based on an agile approach, repeating development and verification in short cycles.

On the other hand, while such iterative development enables rapid improvements, it inherently contains the problem that development environments tend to vary among members. Consequently, dependencies on libraries, OS configurations, build procedures, and execution environments become implicit knowledge. This leads to increased complexity in the development environment across the entire team, resulting in reduced reproducibility and difficulties in handover.

RoboDragons has been developing SSL for many years, and during this time, numerous developers have accumulated software updates. However, some of these updates were made assuming each developer's individual environment, resulting in a decline in the reproducibility of the development environment. This situation increases the startup costs for new members beginning development and can become a factor inducing defects and rework due to environmental differences.

To address the above issues, RoboDragons containerized the development environment using Docker, Docker Compose, and Dev Containers. This approach is expected to enable new members to quickly begin development in an environment shared with existing members, while minimizing discrepancies in environment setup to suppress bugs and rework. Moreover, such problems are not unique to specific teams and are likely to become apparent in many future SSL development environments. Therefore, in this paper, we introduce the concept of containerized development environment setup as one solution to ensure continuity.

2.2 Method for Team Member Onboarding

The containerized development environments consist of Docker, Docker Compose, and Dev Containers. The role of each tool is as follows:

The role of each tool is as follows:

- **Docker**: Virtualizes the operating system and application environment, and defines the specific tools and operating system requirements for *RoboDragonsSystem* (RDS) development through a `Dockerfile`.
- **Docker Compose**: Facilitates the management (startup, shutdown, and networking) of multiple containers, such as RDS and the simulator.
- **Dev Containers**: Provides an integrated develop interface within the container using Visual Studio Code. It supports GUI-based configuration, notifications of configuration updates, and automatic installation of recommended extensions.

In particular, the adoption of Dev Containers enables even new team members with limited experience in command-line operations to launch the development environment through a graphical user interface. Furthermore, by defining the development environment in a `Dockerfile`, we obtained the following benefits:

- **Improved Consistency**: Eliminates variations in operating systems and tool versions, thereby preventing bugs caused by discrepancies between development and production environments.
- **Rapid Updates**: Facilitates the introduction of new tools and system-wide updates by ensuring that all members use an identical environment.
- **Streamlined Engineer Onboarding**: Significantly reduces the time and effort required for new members to set up their initial development environments.

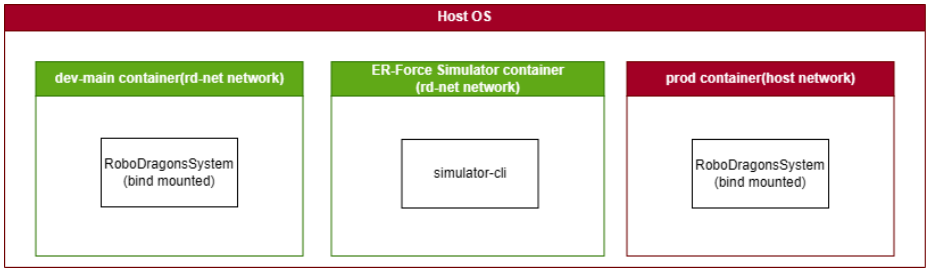


Fig. 2: Configuration of Docker containers.

Fig. 2 illustrates the configuration of our Docker containers. The system consists of three types of containers, —`Development`, `Simulator`, and `Production`—running on a single host operating system.

- **Development**: A container dedicated to the development of RDS.

- **Simulator**: A container for running `simulator-cli`. This container is built from the latest `framework` repository[1] and is published on the *RoboDragons* GitHub Container Registry[2].
- **Production**: A container responsible for real-robot operations.

The network configuration differs depending on whether the system is used for simulation or real-robot operation. When RDS is executed in the simulator environment, an isolated network, `rd-net`, is constructed using Docker's bridge network to prevent interference with external networks. In contrast, for real-robot operations, the `host` network is utilized, as direct access to the host network stack is required for communication with SSL-related software, such as *SSL-Vision*.

To enable seamless source code editing from the host operating system, the RDS directory is bind-mounted into the container. Furthermore, both the `development` and `production` containers are built from the same `Dockerfile`, ensuring that the installed tools and libraries are identical. This consistency enables reliable emergency code modifications within the `production` container, even in time-critical situations such as during a timeout in a match.

The advantages and disadvantages of this container-based operation are summarized as follows:

Advantages

- Unifies the development environment for all developers while increasing the flexibility of the existing toolchain (e.g., migrating the build system from Make to CMake).
- Lowers both psychological and technical barriers to using Docker for developers, including newcomers, by providing a GUI-based interface.

Disadvantages

- **Stability**: Dev Container occasionally exhibits unstable behavior. For instance, synchronization failures may occur between the container's `DISPLAY` environment variable and the host's display number, causing GUI-based applications, such as visualizers, to fail to launch.
- **Offline Operation**: Since building or updating containers requires an internet connection, it is difficult to rebuild the environment under the restricted network conditions typically encountered at competition venues. This limitation necessitates operational rules, such as ensuring that all builds are completed in advance.

In conclusion, while the use of Docker has successfully unified the development environment within the team, addressing these remaining disadvantages remains an important direction for future development.

[1] `https://github.com/robotics-erlangen/framework`
[2] `https://github.com/RoboDragons/framework/pkgs/container/framework`

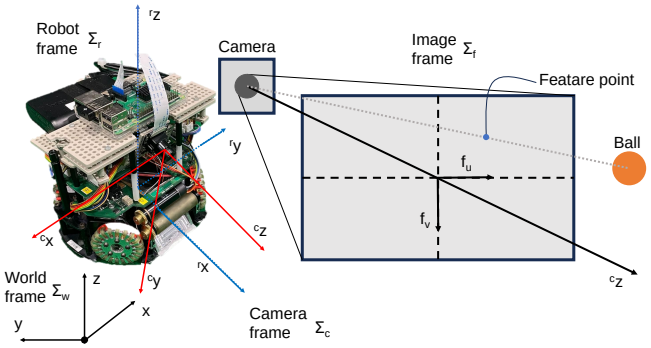


Fig. 3: Frames.

3 Image-Based Visual Servoing via An On-Board Camera

In RoboCup 2019, 2021, and 2022, *SSL-Vision Blackout Challenge* [1–3] were held as one of Technical Challenges in SSL. The goal was to incentivize teams to explore local sensing and processing rather than the typical SSL approach of an off-board computer and a global set of cameras sensing the environment [3]. The challenges were motivated by one of the long term goals—relaxing SSL Vision constraints [4].

Visual feedback control using an on-board camera would be useful for SSL robots even in the main competition using the SSL-Vision. It is because there exists some situation that the ball is temporarily hidden during a game. As the other situation, it would be also affordable when a robot wants to manipulate the ball dexterously, e.g., when dribbling stabilization as presented in our previous ETDP [5].

Our ETDP2020 [6] proposed a simple algorithm for visual feedback control of an SSL robot. The control algorithm does not require any mathematical model of the robot, but does need to switch the lateral/rotational motion and longitudinal motion. This section presents a standard image-based visual servoing (IBVS) controller based on a kinematic model between the robot and camera image, which achieves visual feedback control without switching the motions.

3.1 A Kinematic Model Between The Robot and Image Feature

This subsection derives a kinematic model for IBVS using an SSL robot equipped with an on-board camera. The SSL robot is an omnidirectional mobile robot, and its kinematics differ from those of conventional differential-drive robots. Therefore, it is necessary to derive a kinematic model that integrates the robot kinematics and the camera model. Our local vision system is mainly consisted of a Raspberry Pi and a Pi camera, which is almost the same with in our ETDP2020 [6]. See Ref. [6] for the detail.

Table 1: Definition of symbols.

Symbol	Meaning/Description
$\Sigma_w,\ \Sigma_r$	world and robot frames
$\Sigma_c,\ \Sigma_f$	camera and image frames
$({}^c x, {}^c y, {}^c z)$	a 3D point feature in Σ_c
${}^w\boldsymbol{\alpha}_c = [\phi, \theta, \psi]^\top$	camera's orientation represented by Euler angles in Σ_w
${}^w\boldsymbol{R}_c$	rotation matrix from Σ_c to Σ_w
$\boldsymbol{T}_\omega$	transformation matrix from the time derivative of Euler angles to the angular velocity
$\boldsymbol{f} = [f_u, f_v]^\top$	image feature containing λ, k_i, δ, and f_{i0} $(i = u, v)$
λ	focal length (in meters)
k_i	magnifications in the f_i-axis $(i = u, v)$ (in pixel/meters)
δ	angle between the f_u- and f_v-axes
f_{i0}	the coordinates of the principal point (in pixels)
$\boldsymbol{A}$	transformation matrix that projects $({}^c x/{}^c z, {}^c y/{}^c z, 1)$ to $(\tilde{f}_u, \tilde{f}_v, 1)$ and contains the camera internal parameters: λ, k_i, δ, and f_{i0} $(i = u, v)$
${}^i\boldsymbol{p}_c$	position vector of the camera in Σ_i $(i = w, c)$
${}^i\boldsymbol{p}$	position vector of the ball in Σ_i $(i = w, c)$
${}^i\boldsymbol{v}_c$	camera's linear velocity in Σ_i $(i = w, c)$
${}^i\boldsymbol{\omega}_c$	camera's angular velocity in Σ_i $(i = w, c)$
$[{}^i\boldsymbol{\omega}_c\times]$	skew-symmetric matrix for ${}^i\boldsymbol{\omega}_c$ $(i = w, c)$
d_c	distance along ${}^r x$ from the origin of Σ_r to the origin of Σ_c
h_c	distance along ${}^r z$ from the origin of Σ_r to the origin of Σ_c
$\boldsymbol{q} = [x, y, \phi]^\top$	position and orientation of the robot in Σ_w
$\boldsymbol{v}_{\mathrm{cmd}} = [v_x, v_y, v_a]^\top$	velocity command of the robot
$\boldsymbol{\beta}$	scaling parameters for robot velocity (ratio of the response to command)

Four frames—the world, robot, camera, and image frames—, the symbols used for modeling are shown in Fig. 3 and Table 1. Let us represent an orientation by using Euler angles ${}^w\boldsymbol{\alpha}_c = [\phi, \theta, \psi]^\top$ in this paper. Using two types of basic rotation matrices

$$\boldsymbol{R}_z(\cdot) \coloneqq \begin{bmatrix} \cos(\cdot) & -\sin(\cdot) & 0 \\ \sin(\cdot) & \cos(\cdot) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{R}_y(\cdot) \coloneqq \begin{bmatrix} \cos(\cdot) & 0 & \sin(\cdot) \\ 0 & 1 & 0 \\ -\sin(\cdot) & 0 & \cos(\cdot) \end{bmatrix} \tag{1}$$

defines the rotation matrix ${}^w\boldsymbol{R}_c$ as

$$\begin{aligned} {}^w\boldsymbol{R}_c({}^w\boldsymbol{\alpha}_c) &= \boldsymbol{R}_z(\phi)\,\boldsymbol{R}_y(\theta)\,\boldsymbol{R}_z(\psi) \\ &= \begin{bmatrix} C_\phi C_\theta C_\psi - S_\phi S_\psi & -C_\phi C_\theta S_\psi - S_\phi C_\psi & C_\phi S_\theta \\ S_\phi C_\theta C_\psi + C_\phi S_\psi & -S_\phi C_\theta S_\psi + C_\phi C_\psi & S_\phi S_\theta \\ -S_\theta C_\psi & S_\theta S_\psi & C_\theta \end{bmatrix}, \end{aligned} \tag{2}$$

where $S_{(\cdot)} \coloneqq \sin(\cdot)$ and $C_{(\cdot)} \coloneqq \cos(\cdot)$. Note that this matrix is singular when $\theta = 0$. Also, the time derivative of ${}^w\boldsymbol{\alpha}_c$, i.e., ${}^w\dot{\boldsymbol{\alpha}}_c$, is converted to the angular

velocity ${}^{w}\boldsymbol{\omega}_c$ via

$$ {}^{w}\boldsymbol{\omega}_c = \underbrace{\begin{bmatrix} 0 & -\sin\phi & \cos\phi\sin\theta \\ 0 & \cos\phi & \sin\phi\sin\theta \\ 1 & 0 & \cos\theta \end{bmatrix}}_{\boldsymbol{T}_\omega({}^{w}\boldsymbol{\alpha}_c)} {}^{w}\dot{\boldsymbol{\alpha}}_c. \tag{3} $$

A standard IBVS controller [7, 8] requires a kinematic model between the robot and image feature, which is principally based on two kinds of Jacobians. Those Jacobians are derived from the time derivative of the geometric relationship between the robot and camera and between the camera and image feature. The authors derived the Jacobians with reference to [9] and [10].

The geometric relationship of a point feature between the image plane and 3D space represents as follows:

$$ \begin{bmatrix} f_u \\ f_v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \lambda k_u & -\frac{\lambda k_u}{\tan\delta} & f_{u0} \\ 0 & \frac{\lambda k_v}{\sin\delta} & f_{v0} \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}} \begin{bmatrix} \frac{{}^{c}x}{{}^{c}z} \\ \frac{{}^{c}y}{{}^{c}z} \\ 1 \end{bmatrix}. \tag{4} $$

Suppose that we use the so-called pinhole camera, i.e., a camera with $\delta = \pi/2$ and $f_{u0} = f_{v0} = 0$. Then, Eq. (4) is reduced to

$$ \begin{bmatrix} f_u \\ f_v \end{bmatrix} = \frac{1}{{}^{c}z} \begin{bmatrix} \lambda_u {}^{c}x \\ \lambda_v {}^{c}y \end{bmatrix}, \tag{5} $$

where $\lambda_i \coloneqq \lambda k_i$ $(i = u, v)$. Computing the time derivative of Eq. (5) yields

$$ \begin{bmatrix} \dot{f}_u \\ \dot{f}_v \end{bmatrix} = \begin{bmatrix} \frac{\lambda_u}{{}^{c}z} & 0 & -\lambda_u \frac{{}^{c}x}{{}^{c}z^2} \\ 0 & \frac{\lambda_v}{{}^{c}z} & -\lambda_v \frac{{}^{c}y}{{}^{c}z^2} \end{bmatrix} \begin{bmatrix} {}^{c}\dot{x} \\ {}^{c}\dot{y} \\ {}^{c}\dot{z} \end{bmatrix}. \tag{6} $$

Next, consider the following geometric relationship among Σ_w, Σ_c, and a ball:

$$ {}^{w}\boldsymbol{p} = {}^{w}\boldsymbol{p}_c + {}^{w}\boldsymbol{R}_c {}^{c}\boldsymbol{p} \quad \Leftrightarrow \quad {}^{c}\boldsymbol{p} = {}^{w}\boldsymbol{R}_c^{\top}({}^{w}\boldsymbol{p} - {}^{w}\boldsymbol{p}_c). \tag{7} $$

Differentiating Eq. (7) with respect to time yields

$$ {}^{c}\dot{\boldsymbol{p}} = {}^{w}\dot{\boldsymbol{R}}_c^{\top}({}^{w}\boldsymbol{p} - {}^{w}\boldsymbol{p}_c) + {}^{w}\boldsymbol{R}_c^{\top}({}^{w}\dot{\boldsymbol{p}} - {}^{w}\dot{\boldsymbol{p}}_c). \tag{8} $$

Now assume that a ball is static in Σ_w, i.e., ${}^{w}\dot{\boldsymbol{p}} \equiv \boldsymbol{0}$. Also, note that ${}^{w}\dot{\boldsymbol{R}}_c = [{}^{w}\boldsymbol{\omega}_c\times]{}^{w}\boldsymbol{R}_c$ and thus ${}^{w}\dot{\boldsymbol{R}}_c^{\top} = {}^{w}\boldsymbol{R}_c^{\top}[{}^{w}\boldsymbol{\omega}_c\times]^{\top} = {}^{w}\boldsymbol{R}_c^{\top}[-{}^{w}\boldsymbol{\omega}_c\times]$. Then, Eq. (8) can

be represented as

$$
\begin{aligned}
{}^c\dot{\boldsymbol{p}} &= {}^w\boldsymbol{R}_c^\top[-{}^w\boldsymbol{\omega}_c\times]({}^w\boldsymbol{p}-{}^w\boldsymbol{p}_c) - {}^w\boldsymbol{R}_c^\top{}^w\dot{\boldsymbol{p}}_c \\
&= {}^w\boldsymbol{R}_c^\top[-{}^w\boldsymbol{\omega}_c\times]{}^w\boldsymbol{R}_c \underbrace{{}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{p}-{}^w\boldsymbol{p}_c)}_{{}^c\boldsymbol{p}} - {}^w\boldsymbol{R}_c^\top{}^w\dot{\boldsymbol{p}}_c \\
&= -{}^w\boldsymbol{R}_c^\top{}^w\boldsymbol{\omega}_c\times{}^c\boldsymbol{p} - {}^w\boldsymbol{R}_c^\top{}^w\dot{\boldsymbol{p}}_c = -\underbrace{{}^c\boldsymbol{R}_w{}^w\boldsymbol{\omega}_c}_{{}^c\boldsymbol{\omega}_c}\times{}^c\boldsymbol{p} - \underbrace{{}^c\boldsymbol{R}_w{}^w\dot{\boldsymbol{p}}_c}_{{}^c\dot{\boldsymbol{p}}_c} \\
&= \begin{bmatrix} -1 & 0 & 0 & 0 & -{}^cz & {}^cy \\ 0 & -1 & 0 & {}^cz & 0 & -{}^cx \\ 0 & 0 & -1 & -{}^cy & {}^cx & 0 \end{bmatrix} \begin{bmatrix} {}^c\dot{\boldsymbol{p}}_c \\ {}^c\boldsymbol{\omega}_c \end{bmatrix}.
\end{aligned} \tag{9}
$$

Substituting Eq. (9) into Eq. (6) gives

$$
\begin{aligned}
\begin{bmatrix} \dot{f}_u \\ \dot{f}_v \end{bmatrix} &= \begin{bmatrix} -\frac{\lambda_u}{{}^cz} & 0 & \lambda_u\frac{{}^cx}{{}^cz^2} & \lambda_u\frac{{}^cx\,{}^cy}{{}^cz^2} & -\lambda_u\left(1+\frac{{}^cx^2}{{}^cz^2}\right) & \lambda_u\frac{{}^cy}{{}^cz} \\ 0 & -\frac{\lambda_v}{{}^cz} & \lambda_v\frac{{}^cy}{{}^cz^2} & \lambda_v\left(1+\frac{{}^cy^2}{{}^cz^2}\right) & -\lambda_v\frac{{}^cx\,{}^cy}{{}^cz^2} & -\lambda_v\frac{{}^cx}{{}^cz} \end{bmatrix} \begin{bmatrix} {}^c\dot{\boldsymbol{p}}_c \\ {}^c\boldsymbol{\omega}_c \end{bmatrix} \\
&= \underbrace{\begin{bmatrix} -\frac{\lambda_u}{{}^cz} & 0 & \frac{f_u}{{}^cz} & \frac{f_uf_v}{\lambda_v} & -\left(\lambda_u+\frac{f_u^2}{\lambda_u}\right) & \frac{\lambda_u}{\lambda_v}f_v \\ 0 & -\frac{\lambda_v}{{}^cz} & \frac{f_v}{{}^cz} & \lambda_v+\frac{f_v^2}{\lambda_v} & -\frac{f_uf_v}{\lambda_u} & -\frac{\lambda_v}{\lambda_u}f_u \end{bmatrix}}_{\boldsymbol{J}_f(f_u,f_v,{}^cz)} \begin{bmatrix} {}^c\dot{\boldsymbol{p}}_c \\ {}^c\boldsymbol{\omega}_c \end{bmatrix}.
\end{aligned} \tag{10}
$$

The matrix $\boldsymbol{J}_f$ is called as the *image Jacobian* or *interaction matrix*.

The last part in modeling is to derive a kinematic model between the camera and robot in the 3D space. The linear and angular velocity of the camera in Σ_c can be converted to the one in Σ_w and then to the robot's velocity as follows:

$$
\begin{aligned}
\begin{bmatrix} {}^c\dot{\boldsymbol{p}}_c \\ {}^c\boldsymbol{\omega}_c \end{bmatrix} &= \begin{bmatrix} {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} {}^w\dot{\boldsymbol{p}}_c \\ {}^w\boldsymbol{\omega}_c \end{bmatrix} \\
&= \begin{bmatrix} {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{T}_\omega({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} {}^w\dot{\boldsymbol{p}}_c \\ {}^w\dot{\boldsymbol{\alpha}}_c \end{bmatrix} \\
&= \begin{bmatrix} {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{T}_\omega({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} \frac{\partial\,{}^w\boldsymbol{p}_c}{\partial\boldsymbol{q}}\dot{\boldsymbol{q}} \\ \frac{\partial\,{}^w\boldsymbol{\alpha}_c}{\partial\boldsymbol{q}}\dot{\boldsymbol{q}} \end{bmatrix}.
\end{aligned} \tag{11}
$$

The robot's kinematic equations of motion can be written as

$$
\dot{\boldsymbol{q}} = \boldsymbol{R}_z(\phi)\,\boldsymbol{\beta}\,\boldsymbol{v}_{\mathrm{cmd}}, \tag{12}
$$

where the only difference with Ref. [10] is to add scaling parameters. Therefore, substituting Eq. (12) into Eq. (11) brings

$$
\begin{bmatrix} {}^c\dot{\boldsymbol{p}}_c \\ {}^c\boldsymbol{\omega}_c \end{bmatrix} = \underbrace{\begin{bmatrix} {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & {}^w\boldsymbol{R}_c^\top({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{T}_\omega({}^w\boldsymbol{\alpha}_c) \end{bmatrix} \begin{bmatrix} \frac{\partial\,{}^w\boldsymbol{p}_c}{\partial\boldsymbol{q}} \\ \frac{\partial\,{}^w\boldsymbol{\alpha}_c}{\partial\boldsymbol{q}} \end{bmatrix} \boldsymbol{R}_z(\phi)\,\boldsymbol{\beta}}_{\boldsymbol{J}_c({}^w\boldsymbol{\alpha}_c)}\,\boldsymbol{v}_{\mathrm{cmd}}, \tag{13}
$$

where the forward kinematics

$$ {}^{w}\boldsymbol{p}_c \coloneqq \begin{bmatrix} {}^{w}x_c \\ {}^{w}y_c \\ {}^{w}z_c \end{bmatrix} = \begin{bmatrix} x + d_c \cos\phi \\ y + d_c \sin\phi \\ h_c \end{bmatrix} \tag{14} $$

and Euler angles gives the following gradients:

$$ \frac{\partial\, {}^{w}\boldsymbol{p}_c}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial {}^{w}x_c}{\partial q_1} & \frac{\partial {}^{w}x_c}{\partial q_2} & \frac{\partial {}^{w}x_c}{\partial q_3} \\ \frac{\partial {}^{w}y_c}{\partial q_1} & \frac{\partial {}^{w}y_c}{\partial q_2} & \frac{\partial {}^{w}y_c}{\partial q_3} \\ \frac{\partial {}^{w}z_c}{\partial q_1} & \frac{\partial {}^{w}z_c}{\partial q_2} & \frac{\partial {}^{w}z_c}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d_c \sin\phi \\ 0 & 1 & d_c \cos\phi \\ 0 & 0 & 0 \end{bmatrix}, \tag{15} $$

$$ \frac{\partial\, {}^{w}\boldsymbol{\alpha}_c}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial \phi}{\partial q_1} & \frac{\partial \phi}{\partial q_2} & \frac{\partial \phi}{\partial q_3} \\ \frac{\partial \theta}{\partial q_1} & \frac{\partial \theta}{\partial q_2} & \frac{\partial \theta}{\partial q_3} \\ \frac{\partial \psi}{\partial q_1} & \frac{\partial \psi}{\partial q_2} & \frac{\partial \psi}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{16} $$

Note that $\boldsymbol{J}_c$ is independent of $\boldsymbol{q}$ similarly to the result as shown in Ref. [9].

Finally, combining Eq. (13) with Eq. (10) provides the following velocity relationship between the velocity command to the robot and the motion of an image feature:

$$ \dot{\boldsymbol{f}} = \underbrace{\boldsymbol{J}_f(\boldsymbol{f}, {}^{c}z)\, \boldsymbol{J}_c({}^{w}\boldsymbol{\alpha}_c)}_{\boldsymbol{J}_{\mathrm{vis}}(\boldsymbol{f}, {}^{c}z, {}^{w}\boldsymbol{\alpha}_c)}\, \boldsymbol{v}_{\mathrm{cmd}}. \tag{17} $$

3.2 IBVS controller

To solve Eq. (17) with respect to $\boldsymbol{v}_{\mathrm{cmd}}$, the inverse of $\boldsymbol{J}_{\mathrm{vis}}$ cannot be used because $\boldsymbol{J}_{\mathrm{vis}}$ is a 2×3 matrix. Instead, consider to solve the following minimization problem:

$$ \min_{\boldsymbol{v}_{\mathrm{cmd}}} \| \dot{\boldsymbol{f}} - \boldsymbol{J}_{\mathrm{vis}}\, \boldsymbol{v}_{\mathrm{cmd}} \|. \tag{18} $$

We can obtain the general solution

$$ \boldsymbol{v}^{\star}_{\mathrm{cmd}} = \boldsymbol{J}^{+}_{\mathrm{vis}}(\boldsymbol{f}, {}^{c}z, {}^{w}\boldsymbol{\alpha}_c)\, \dot{\boldsymbol{f}} + \boldsymbol{J}^{\perp}_{\mathrm{vis}}(\boldsymbol{f}, {}^{c}z, {}^{w}\boldsymbol{\alpha}_c)\, \boldsymbol{\epsilon}, \tag{19} $$

where $\boldsymbol{J}^{+}_{\mathrm{vis}} = \boldsymbol{J}^{\top}_{\mathrm{vis}}(\boldsymbol{J}_{\mathrm{vis}}\boldsymbol{J}^{\top}_{\mathrm{vis}})^{-1} \in \mathbb{R}^{3\times 2}$ denotes the pseudo-inverse of $\boldsymbol{J}_{\mathrm{vis}}$, $\boldsymbol{J}^{\perp}_{\mathrm{vis}} = \boldsymbol{I}_3 - \boldsymbol{J}^{+}_{\mathrm{vis}}\boldsymbol{J}_{\mathrm{vis}} \in \mathbb{R}^{3\times 3}$ is the orthogonal projection operator into the null-space of $\boldsymbol{J}_{\mathrm{vis}}$, i.e., $\ker \boldsymbol{J}_{\mathrm{vis}}$, and $\boldsymbol{\epsilon} \in \mathbb{R}^3$ is an arbitrary vector. There exists an infinite number of solutions in Eq. (19) for arbitrary $\boldsymbol{\epsilon}$. Note that a solution $\boldsymbol{v}^{\star}_{\mathrm{cmd}}$ which satisfies $\min_{\boldsymbol{v}_{\mathrm{cmd}}} \|\boldsymbol{v}_{\mathrm{cmd}}\|$ in addition to Eq. (18) is equal to the one that $\boldsymbol{\epsilon} = \boldsymbol{0}_3$ in Eq. (19).

To drive $\boldsymbol{f}$ into a desired image feature $\boldsymbol{f}^{\star}$, we design $\dot{\boldsymbol{f}}$ as a proportional (P) feedback

$$ \dot{\boldsymbol{f}}(t) = \boldsymbol{\kappa}_P(\boldsymbol{f}^{\star} - \boldsymbol{f}(t)) \tag{20} $$

or a proportional-integral (PI) feedback

$$ \dot{\boldsymbol{f}}(t) = \boldsymbol{\kappa}_P(\boldsymbol{f}^{\star} - \boldsymbol{f}(t)) + \boldsymbol{\kappa}_I \int_0^t (\boldsymbol{f}^{\star} - \boldsymbol{f}(\tau))\mathrm{d}\tau, \tag{21} $$

where $\boldsymbol{\kappa}_P = \mathrm{diag}\{\kappa_{Pu}, \kappa_{Pv}\}$ and $\boldsymbol{\kappa}_I = \mathrm{diag}\{\kappa_{Iu}, \kappa_{Iv}\}$ are gain matrices. Implementing Eqs. (19) and (20) or Eqs. (19) and (21) enables the robot to execute a IBVS task.

3.3 Experimental Evaluation

The derived IBVS controller (19), (21) is validated via an experiment using a RoboDragons 7G robot. The robot equips a local vision system composed of a Raspberry Pi and a Pi camera.

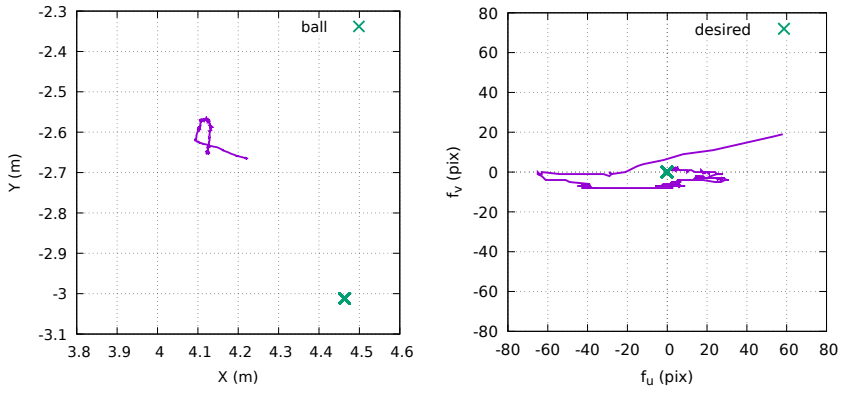
The image resolution of the camera were 320×240 pixels; the frame rate of the camera was 30 fps, i.e., the image data was updated every 33.3 ms. The captured image was binarized to extract the barycentric coordinates of each marker's area as the coordinates of a point feature. The coordinates of the point feature were sent to our host PC via socket networking.

On the host PC, using the coordinates of the point feature, velocity commands based on a set of (19) and (21). The velocity commands were sent to the robot via wireless communication at every 17.6 ms. Feedback control to the commands by the embedded system drove the robot.

The parameters with respect to the robot and camera were as follows:

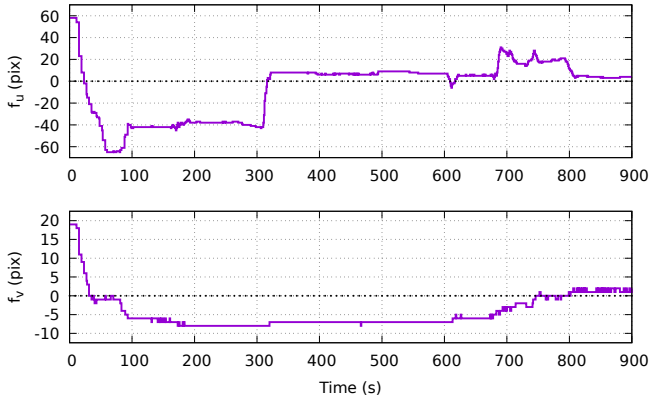
- $\lambda_x = 490.7\,\mathrm{pixels}$ and $\lambda_y = 426.7\,\mathrm{pixels}$;
- $\phi = 0\,\mathrm{rad}$, $\theta = (11/18)\pi\,\mathrm{rad}$, and $\psi = -\pi/2\,\mathrm{rad}$;
- $d_c = 0.09\,\mathrm{m}$ and $h_c = 0.05\,\mathrm{m}$;
- $\boldsymbol{\beta} = \mathrm{diag}\{0.9698518993, 0.9132887678, 0.9482374625\}$.

The coordinates of the desired feature $\boldsymbol{f}^\star$ were set to $(0, 0)$, i.e., the center of the image plane. Letting the depth be a constant value in the desired configuration, we set ${}^c z = 0.3\,\mathrm{m}$.

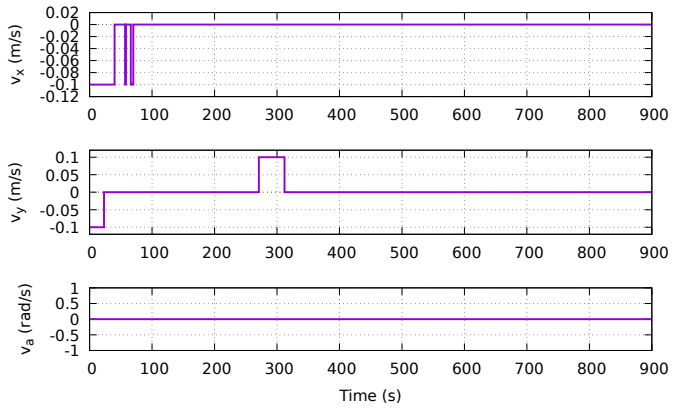


(a) on the field (b) on the image plane

Fig. 4: Trajectories.



(a) image feature coordinates



(b) $\boldsymbol{v}_{\mathrm{cmd}}$

Fig. 5: Time plots.

Figures 4 and 5 show an experimental result with the following gains: $\kappa_{Pu} = \kappa_{Pv} = 2$ and $\kappa_{Iu} = \kappa_{Iv} = 0.5$. In Fig. 4, (a) depicts the trajectory of the center of the robot on the field, and (b) shows the trajectory of the point feature on the image plane (i.e., Σ_f). It can be seen that backward and forward motion of the robot drove the point feature to the desired one. Figures 5 (a) and (b) provide the time responses of $\boldsymbol{f}$ and $\boldsymbol{v}_{\mathrm{cmd}}$, respectively. These graphs represent that varying velocity commands slightly yielded that the point feature converged to zero with some overshoot.

4 Concluding Remarks

This paper has shared the technical updates of RoboDragons 2025–2026 with the SSL. The contents are composed of two parts: the first is about the improvement of team member onboarding by using Docker tools and the second is about modeling and controller design for IBVS.

Acknowledgement.

This work was partially supported by JSPS KAKENHI Grant Number 23K11338, The Nitto Foundation, and Aichi Prefectural University Alumni Association. The authors would like to thank Yamashita, S. and Matsubara, H. (e-Valley, Co. Ltd., Japan) for their technical support. The authors also would like to thank the other active members of RoboDragons 2025–2026.

References

1. RoboCup Soccer Small Size League Technical Committee: "SSL-Vision Blackout Technical Challenge," 2019-01-03; `https://robocup-ssl.github.io/technical-challenge-rules/2019-ssl-vision-blackout-rules.pdf` (accessed on 26 Jan 2026)
2. RoboCup Soccer Small Size League Technical Committee: "Vision Blackout Technical Challenge Rules," 2021-06-23; `https://robocup-ssl.github.io/technical-challenge-rules/2021-ssl-vision-blackout-rules.pdf` (accessed on 26 Jan 2026)
3. RoboCup Soccer Small Size League Technical Committee: "Vision Blackout Technical Challenge Rules," 2022-06-15; `https://robocup-ssl.github.io/technical-challenge-rules/2022-ssl-vision-blackout-rules.pdf` (accessed on 26 Jan 2026)
4. RoboCup Soccer Small Size League Technical Committee: "Principles and Goals," 2020-12-14; `https://robocup-ssl.github.io/ssl-goals/sslgoals.html` (accessed on 26 Jan 2026)
5. Ito, M., Shibata, M., and Shimizu, T.: "RoboDragons 2025 extended team description," RoboCup Soccer Small Size League, 2025. Available online: `https://ssl.robocup.org/wp-content/uploads/2025/04/2025_ETDP_RoboDragons.pdf` (accessed on 26 Jan 2026).
6. Ito, M., Nakayama, M., Ando, Y., Adachi, Y., Du, J., Suzuki, R., Ono, Y., Kashiwamori, F., Ban, K., Isokawa, S., Yamada, T., and Naruse, T.: "RoboDragons 2020 extended team Description," RoboCup Soccer Small Size League, 2020. Available online: `https://ssl.robocup.org/wp-content/uploads/2020/03/2020_ETDP_RoboDragons.pdf` (accessed on 13 Feb 2024).
7. Chaumette, F., Hutchinson, S.: "Visual servo control: I. Basic approaches," IEEE Robotics & Automation Magazine, 13(4), 82–90 (2006)
8. Spong, M.W., Hutchinson, S., and Vidyasagar, M.: "Robot Modeling and Control," 1st Edition, Wiley (2006)
9. De Luca, A., Oriolo, G., Giordano, P.R.: "Image-based visual servoing schemes for nonholonomic mobile manipulators: theory and experiments," Robotica, 25(10), 131–145 (2007)

10. Barreto, J.C.L., Conceição, A.G.S., Dórea, C.E.T., Martinez, L., de Pieri, E.R.: "Design and implementation of model-predictive control with friction compensation on an omnidirectional mobile robot," IEEE/ASME Transactions on Mechatronics, 19(2), 467–476 (2014)