Tracking and Closed-Loop Control of HeRo Robots

Varun Raveendra
Department of Electrical and Computer Science Engineering
University of Utah, Salt Lake City, USA
varun.raveendra@utah.edu

Abstract

Robot tracking and localization pose challenges in swarm robotics, particularly in expansive areas with limited sensor capabilities. This project addresses these challenges by implementing a novel webcam tracking mechanism. Additionally, it provides a user-friendly GUI for setting navigation points. This project serves as a base for analyzing swarm robotics behaviors. This process is demonstrated on HeRo, a low-cost robot developed by VerLab, PhD students from Brazil. This initiative contributes to advancing swarm robotics by tracking and enabling intuitive user interaction for applications in diverse environments.

1 Introduction

This project builds on a particular area of mobile robots that can be integrated into swarm robotics. These robots can be used in multiple cases, including research and teaching. This framework can be used to understand centralized or decentralized control and their behaviors. One of the challenges in mobile robots that are small and limited by their sensing capabilities is the concept of localization. Although the Robots used in a swarm have the sensors used for a robot to localize, given their dimensions compared to their deployment area, it would take significant movement and time to localize. Using sensors like LIDAR would significantly increase the cost, defeating the low-cost solution for swarm robots, and would be inappropriate given the dimensions of the robots we are trying to maintain.

This project serves as a proof of concept for using small robots with limited sensing capabilities to test and verify swarm behaviors' capabilities. Using HeRo robots made by VERLab [Rezeck, Azpurua, Correa, and Chaimowicz (2022)] is the most cost-effective solution for getting started with swarm robotics compared to its competitor, E-Puck. While E-Puck retails at $970, it would take only $70 to build the HeRo robots. In this report we will go through the evolution process the HeRo robot has been through to match our requirements and conclude with the future changes and additions that would better improve the capabilities of the HeRo Robot. [Verlab (2022)]

2 Background Work

To minimize the cost of these swarm robots, a team of PhD students at VERLab in Brazil designed it as an open-source platform to use swarm robots for research. They showcase the robot's capabilities with IMU and the results are better than the E-Puck robot. Since then, the HeRo robot has been modified and updated several times by the members of VERLab. The build instructions, materials, and source code are made freely available in VERLab's GitHub repository. The team demonstrated the use of HeRo robot for various applications like Cooperative transportation, Flock Behaviour, SLAM/Gmapping and Decentralized control.

Using this repository [Verlab (2022)] a team from DRL (Drew Research Lab) at the University of Utah, incorporated these HeRo robots to solve the problem of communication between swarm robots using high-frequency sound waves. The team aimed at using a sound source localization for swarm interactions. There were significant changes that the team made while incorporating the HeRo robots for their project, while some of the changes gave positive results some of the changes did not yield the behaviour of the robots that was required. The team had made changes to the code by writing their own controller code, the encoder shaft was modified to prevent it from breaking due to friction and some

parts were substituted to match the requirements for the robots. The encoder shaft was still an issue and was not being used which made the motors of the robot running an open-loop control mechanism. This method is prone to errors and does not provide good control; this is critical in terms of differential drive robots.

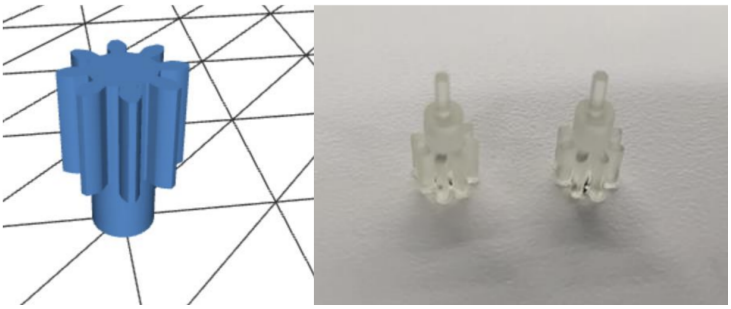


Figure 1: Version 2 encoder shaft (VERLab) vs Version 3 encoder shaft

The concept of tracking the robot using an AR tracker and a webcam was also incorporated but did not work as expected due to the errors in the source code and the ROS1 framework integration.

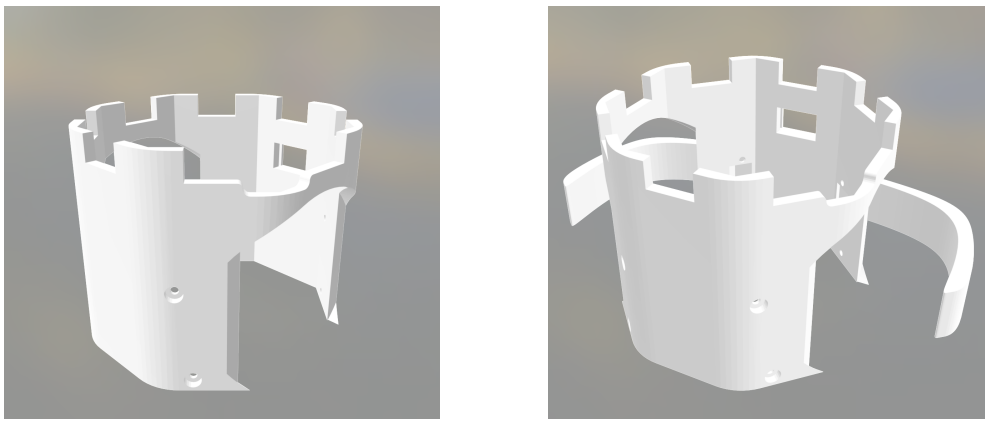
3 Solution

This project addresses the issues that the previous team faced while working with the HeRo robots. Also, it integrates a seamless user interaction with the robots using a Graphic Interface.

3.1 Redesigned Encoder Placement

Encoders are essential for motor control, ensuring the velocity errors are controlled and minimized. Therefore it was important to bring back the encoders into the system. The placement of encoders had no issues mentioned by the makers at VERLab, but it had lots of issues and was unstable. The shaft designed by VERLab [Rezeck et al. (2022)] had the gear attached at the end, which is aligned with the inner circumference of the wheel, and had the indentations for the gear to be placed. In most cases, the shaft broke due to friction and slipping of the gears.

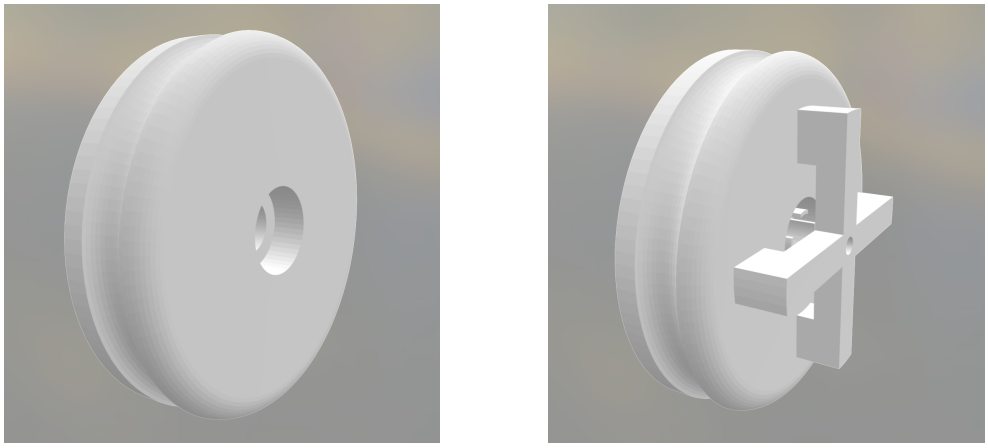
The initial approach of incorporating encoders into the HeRo robots by printing and testing multiple iterations of the encoder shaft proved to be an ineffective solution for ensuring durability. To address this issue, an alternative model was tested to house the encoders more effectively. Figure 2 illustrates the robot's body before and after the implementation of this change. We can observe an additional "arm-like" structure on the robot, which serves as a protective housing for the encoders. These encoders are strategically placed outside the robot, right above the wheels, which feature the structure shown in Figure 3.



(a) Without Encoder Holder (b) With Encoder Holder

Figure 2: Body Modifications

The wheel is connected to a rotating shaft, and the encoder is attached to this shaft. The non-rotating end of the encoder, with its pins, is attached to the "arm-like" structure, preventing the encoder from rotating along with the shaft. This modification changes the rotation scale from six encoder rotations per wheel rotation to one encoder rotation per wheel rotation, resulting in fewer ticks before the wheel completes a full rotation. To accommodate this change, adjustments were necessary in the source code when calculating the robot's distances and velocity. Specifically, the motor step distance parameter in the config.h file under the firmware directory [Verlab (2022)] had to be updated to reflect the new encoder rotation scale accurately.



(a) Without Encoder attachments (b) With Encoder attachments

Figure 3: Wheel Modifications

Overall, the new encoder placement design offers a durable and fixed solution, minimizing the room for errors and ensuring reliable and accurate encoder readings for the HeRo robots' navigation and control systems.

3.2 Re-coding the Velocity Controller and tuning the control loop

Due to the change in the encoder placement, it became necessary to re-tune the PID (Proportional-Integral-Derivative) values to accommodate this modification. During the process of re-tuning these values, it was discovered that the PIDv1 library [Arduino (2022)], which was previously used from the Arduino library, had compatibility issues with the new encoder placement, to mitigate this issue, a new velocity controller code was developed and incorporated into the HeRo robots.

The control loop in the new velocity controller is now tuned using only Proportional and Derivative (PD) values. It was observed that the Integral value for the right motor accumulated errors in the opposite direction, leading to potential instabilities. To address this, the Integral factor was set to zero, effectively eliminating the integral component from the control loop. However, the option to introduce the Integral factor remains available for future fine-tuning, should it be deemed necessary. By developing a custom velocity controller code and tuning the control loop with PD values, the HeRo robots' motion control system was effectively optimized for the new encoder placement. This approach ensures better velocity control, minimizes errors, and enhances the overall stability and responsiveness of the robots' movements, further improving their performance.

3.3 Robot Tracking - Position Estimation

To enhance odometry calculations and improve localization for the HeRo robots in large environments, we integrated webcam tracking as a supplementary solution. The previous team encountered challenges with tracking the HeRo robots using an AR tracker, so we explored alternative approaches and decided to employ simple image processing techniques. The existing sensor system had limitations in spacious and empty areas, so we turned to computer vision techniques for a more accurate estimate of the robot's position.

By utilizing the OpenCV2 library, we process the webcam images in several steps. First, we convert the image to grayscale, and then apply a threshold function to detect the white robots, creating a binary image. The detected robot is then outlined with a bounding box, and because we know the physical dimensions of our robot, we can calculate its position by measuring the number of pixels within the bounding box. This method provides an effective way to estimate the robot's position within the environment. [Daher (2016)]

pixels_per_metric = object_width_pixels / know_robot_width_metric
unknown_object_width= unknown_object_width_pixels / pixels_per_metric

In the ROS framework, the webcam is employed to generate a continuous stream of frames, which are then published for real-time image processing. Through this process, we are able to determine the position estimate of the robot with respect to its surroundings.

Figure 4: Raw Webcam input stream

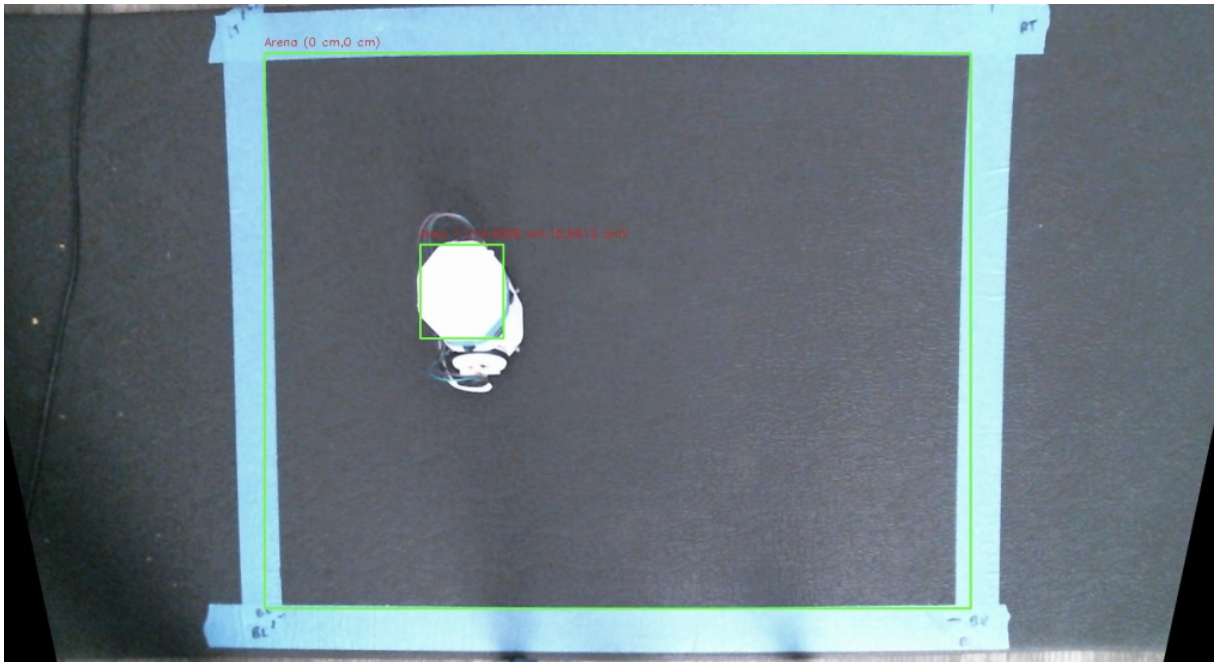


Figure 5: Webcam stream after image processing

For tests, we positioned the webcam on a tripod to provide an aerial view of the robots. While this setup introduces parallax error [Tian, Kyte, and Messer (2002)] due to the camera placement, we have accounted for this issue and applied image processing techniques to mitigate its effects. Overall, this approach has proven valuable to our localization system, offering improved performance in expansive environments.

3.4 GUI Control

To enable seamless user-robot interaction, a new graphical user interface (GUI) was designed. This interface eliminates the need for a teleop keyboard, allowing users to intuitively control the robot through a display window and mouse clicks. When a user clicks on a location in the image, the GUI node processes the input and translates it into the corresponding real-world coordinates. This destination information is then sent to the navigation node for processing.

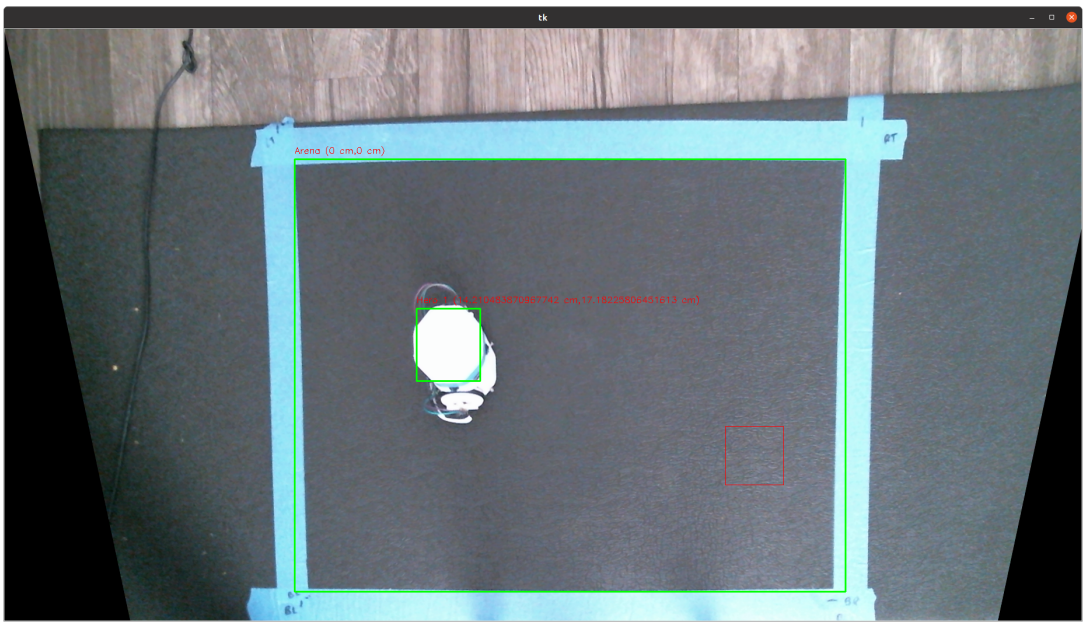


Figure 6: GUI Window

The navigation node utilizes camera and IMU sensor data to determine the robot's current position and orientation. It then calculates the required orientation and position to reach the desired location and sends velocity commands to the robot once the conditions are met. The robot moves along a straight path to reach the target location. The process of orienting the robot to face the desired position relies on mathematical calculations to ensure the shortest and fastest path. The algorithm works as follows:

1. The IMU sensors provide data, which is converted to degrees and normalized to ensure consistency in sign.
2. The difference between the current and required orientations is calculated, and based on the result's magnitude and sign, the robot's rotation direction is determined.
3. The IMU sensors continuously monitor the robot's orientation, ensuring accurate alignment before forward movement is initiated.

This closed-loop control system enables users to effortlessly interact with the robot, while the robot efficiently and accurately navigates to the desired location.

4 Results

The integration of the solutions mentioned above into the system has been meticulously tested and refined through multiple iterations of program updates. The various components, such as the camera feed and the graphical user interface (GUI), which allows a user to designate the desired destination within

the recognized arena, have been thoroughly tested to ensure smooth operation. The current position of the robot and the arena are highlighted with green rectangles, while the destination is indicated by a red box when the user clicks on the camera feed window. Additionally, the robot's location within the arena is captioned.

The control node, which is responsible for issuing velocity commands to the robot's wheels, has been integrated to work seamlessly with the navigational node. The navigational node processes the sensor data and the desired destination, calculates the appropriate velocity commands, and ensures that the robot reaches its goal. The process of integrating these nodes is visualized in the following figure:

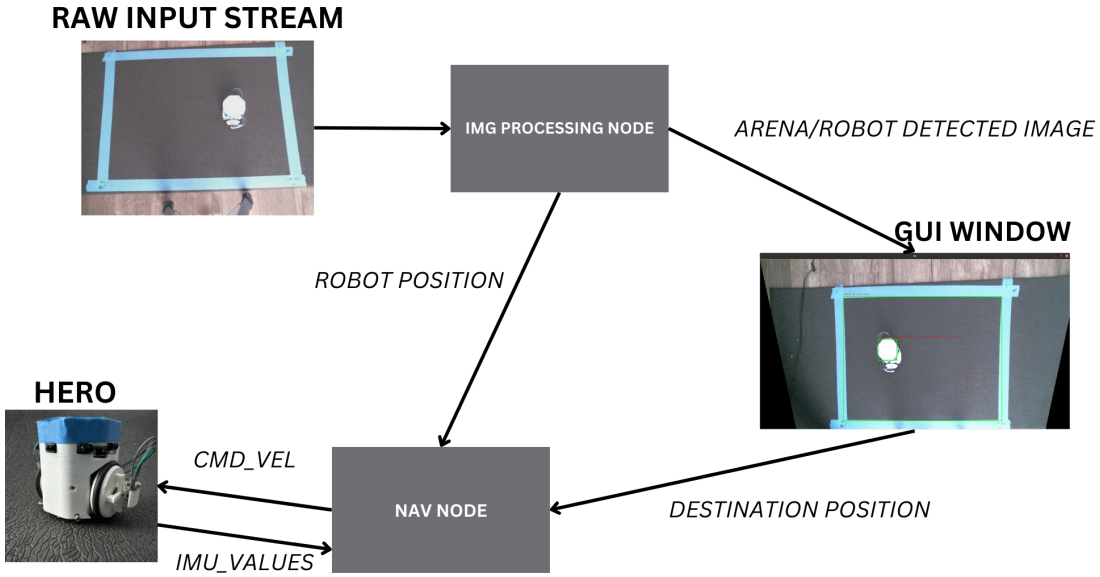


Figure 7: System working overview

The camera and GUI nodes are initialized first, followed by the startup of the ROS serial server to establish communication with the HeRo robot. The GUI node provides the user with a visual representation of the robot's position and allows them to set the destination. This information is then relayed to the navigational node, which in turn controls the velocity controllers.

While the robot's movement can be further optimized, the same applies to the GUI, which currently exhibits a flickering effect. These issues can be addressed in the future by fine-tuning the system and possibly by leveraging different open-source software to minimize the flickering and enhance performance.

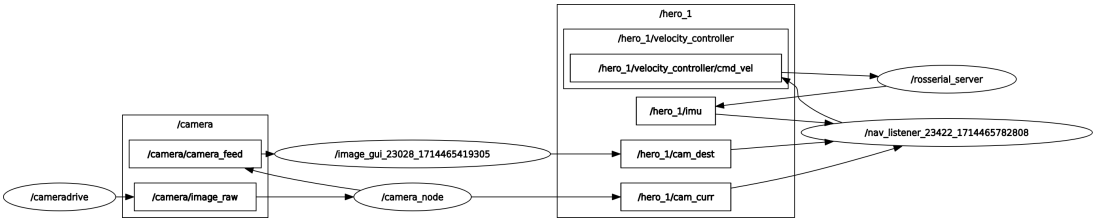
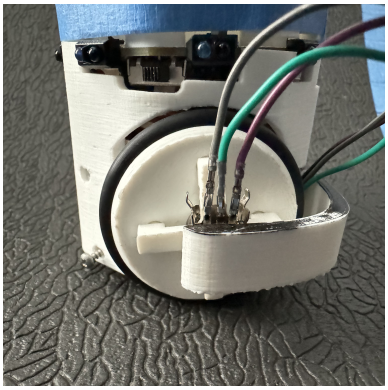
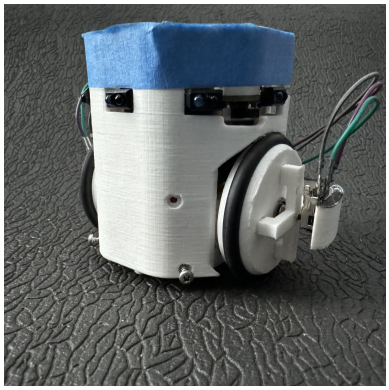


Figure 8: RQT_Graph

The camera node is designed to recognize the arena by detecting rectangular shapes on the floor. However, this does not impede the robot's functionality or control. The camera does not need to be stationary; it can move while still monitoring the robot as long as the robot remains within its field of view. This opens up possibilities for dynamic destination positioning or even for the robot to follow the camera as it moves.



(a) Encoder Setup



(b) Modified HeRo Robot

Figure 9: Images of HeRo Robot

5 Limitations and Future Work

The construction of the robot body through 3D printing is a cost-effective method but does require specialized expertise. The current design includes parts that are suitable for educational purposes but may not be scalable for mass production. The use of higher-quality components could enhance the robustness and performance of the robot.

The webcam-based detection system is reliant on sufficient lighting conditions and can experience false positives due to over-exposure [Guo, Cheng, Zhuo, and Sim (2010)]. This limitation can be addressed in the future by incorporating machine learning algorithms that are more robust to varying lighting conditions and can accurately discern between real robots and potential reflections or shadows. Machine learning could also be employed to compensate for parallax errors, which are inherent in traditional image processing techniques.

The current velocity controller, which is tuned using a proportional-derivative (PD) control strategy, could be further optimized by incorporating an integral term. This adjustment could potentially improve the control precision, but if it leads to degraded performance, alternative control schemes or motor types, such as DC motors, could be considered. DC motors offer better control and might be a quieter alternative to the continuous servo motors currently used.

To enhance the range detection capabilities, the robot could be equipped with Time-of-Flight (ToF) sensors [SMARTlab-Purdue (2022)], which have a significantly longer range than traditional IR sensors. ToF sensors can measure distances up to 5 meters, compared to the approximately 20-centimeter range of IR sensors. This upgrade would significantly improve the robot's ability to localize itself and extend the range of detection, which is particularly important for swarm robotics applications.

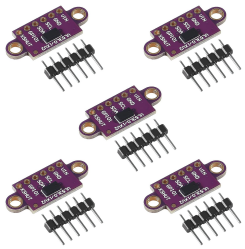


Figure 10: TOF Sensors

References

Paulo Rezeck, Hector Azpurua, Mauricio FS Correa, and Luiz Chaimowicz. Hero 2.0: A low-cost robot for swarm robotics research, 2022.

Verlab. Hero resources. https://github.com/verlab/hero_common/tree/noetic-devel/hero_resources/media/images, 2022.

Arduino. Pid library. https://www.arduino.cc/en/libraries/pid_v1_bc/, 2022.

Adrian Daher. Measuring size of objects in an image with opencv. *PyImageSearch*, March 2016. URL https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/.

Zong Tian, Michael Kyte, and Carroll Messer. Parallax error in video-image systems. *Journal of Transportation Engineering*, 128:218, 05 2002. DOI: 10.1061/(ASCE)0733-947X(2002)128:3(218).

Dong Guo, Yuan Cheng, Shaojie Zhuo, and Terence Sim. Correcting over-exposure in photographs. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 515–521, 2010. DOI: 10.1109/CVPR.2010.5540170.

SMARTlab-Purdue. Smartmbot repository. https://github.com/SMARTlab-Purdue/SMARTmBOT/tree/main/smartmbot_controller_pkg/smartmbot_controller_pkg, 2022.