4-24-2023

Capture the Flag Robot

Joshua Wade Fry

Christa Hope Manges

Capture the Flag Robot

by

Joshua Wade Fry and Christa Hope Manges

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

April 24, 2023

Honors Capstone Director: Mr. Adam Colwell

Part-Time Lecturer of Computer Science

Joshua W Fry 04/24/2023
Student Date

Christa Hope Manges 04/24/2023
Student Date

Adam Colwell 4/25/2023
Director Date

Letha Etzkorn Digitally signed by Letha Etzkorn
Date: 2023.04.29 07:26:55 -05'00'
Department Chair Date

William Wilkerson Digitally signed by William Wilkerson
Date: 2023.05.01 12:18:02 -05'00'
Honors College Dean Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Joshua Wade Fry______________

Student Name (printed)

____

Student Signature

04/24/2023___

Date

Christa Hope Manges___________

Student Name (printed)

Christa Hope Manges
____

Student Signature

04/24/2023___

Date

Table of Contents

Abstract

Throughout the course of the Spring 2023 semester, Joshua Fry and Christa Manges made progress in coding a robot to play Capture the Flag. The project is part of the INCLUDE program, in which students from different academic majors work on a project together. The coding for this semester focused on image processing and robot motion. For image processing, a color image and depth image come from a Kinect sensor. Then objects are recognized in the color image. Information about the objects recognized and their locations is synchronized with the corresponding depth image. Then the information is combined into unified messages to be sent on to other parts of the system. One of these messages is sent to a script controlling the motion of the robot. The robot can then maneuver through the course to successfully play a game of Capture the Flag.

Introduction

Go! Go! Get the flag! Capture the flag is a well-known game consisting of two teams where each team tries to steal the other team's flag and bring it back to base. But what if robots could play capture the flag? This semester, Christa Manges and Joshua Fry joined the INCLUDE program, in which team members from different academic majors come together to complete a project. Our responsibility was to help code a Clearpath Husky robot (see A1) to play capture the flag. In order for a robot to play capture the flag, it must be able to detect what objects are around itself. The robot must also be able to move toward the flag, away from the enemy robot, and around obstacles. Throughout this semester, we have implemented object detection and robot motion.

Process

At the beginning of the semester, we began implementing object recognition with ImageAI. We wanted to make a test script to explore how image recognition works. ImageAI is a Python library that recognizes and detects the location of objects in an image (Olafenwa). We used it with an algorithm called YOLO. YOLO stands for You Only Look Once and is a prominent lightweight object recognition algorithm (Keita). We first used ImageAI with a laptop webcam and later used it with a Kinect sensor. However, we soon learned that ROS (Robot Operating System) is middleware and not its own programming language. We then decided to use ROS and therefore to use an object recognition tool that would more easily integrate into ROS.

Coming into this project at the beginning of the semester, we were fully unaware of what ROS was and how it worked. We were told by other members of the team that it was a proprietary language that the robot utilized to process any inputs faster than other languages. Knowing that the robot could also run Python and C++ scripts, we decided initially to work solely in Python for the sake of ease. After speaking with Dr. Howard Chen, we learned more about ROS and realized our idea of it was entirely incorrect. ROS stands for Robotic Operating System, and it is a framework that runs on Linux. The languages of choice for ROS are C++ and Python, and any scripts that run on ROS can interact with each other regardless of language. Knowing this, we realized very quickly that ROS, along with nodes that would be supported by both C++ and Python, was what we wanted to use for this project.

One element of ROS that is imperative to learn and understand before any progress can be made on a project is how nodes will interact with each other. To do this, we created a diagram representing the different nodes that will run on the robot and the inputs and outputs flowing between them (see A2). Though not all nodes have been created, those that have been and those that will be in the future are both included. The first node to provide input to others is iai_kinect2_opencv4, whose outputs are taken as input by both darknet_ros and image_processor. The output from darknet_ros is read by image_processor and the yet to be created tablet_control. Three nodes read different outputs from image_processor: launcher_processor, arm_processor, and movement_processor. Out of these three nodes, movement_processor is the only one which has been created for this semester. The output from movement_processor is read by the built-in husky node, which allows the robot to move. The output from launcher_processor is read by both launcher_control and tablet_control, the latter of which can also output back to launcher_processor. The output from arm_processor is read by arm_control to manipulate the robotic arm on the robot. Finally, output from tablet_control is also read by the husky node. Thanks to ROS, the interaction between all of these nodes is handled on an individual basis, and each node can be developed independently.

Once we decided we were using ROS, we needed to find a way to integrate the Kinect sensor into ROS and perform object recognition with the resulting image. In order to connect the Kinect sensor into ROS, we used a pre-built ROS node called iai_kinect2_opencv4. The node takes input from a connected Kinect sensor and outputs both camera images and lidar depth images (Paul). Specifically, we use information from the /kinect2/hd/image_color_rect topic for the color image (see A3 for an example of output) and information from the /kinect2/hd/image_depth_rect topic for the depth (distance) image (see A4 for an example of output). Christa calibrated the Kinect sensor by taking pictures of a checkerboard image so that the image_depth_rect topic would output more precise values (see A5 for the setup). The color image then goes to another prebuilt node, darknet_ros, for object detection. darknet_ros uses YOLO to identify objects in the image and determines their locations within the image. This information is output on the /darknet_ros/bounding_boxes topic (Bjelonic). Christa made two modifications to this prebuilt node. First, she modified the code to make the timestamp on the message on /darknet_ros/bounding_boxes to be the same as the timestamp on the corresponding image message on the /kinect2/hd/image_color_rect. If the timestamp does not change while it passes through the darknet_ros node, it enables easier synchronization later on between the output of the darknet_ros node and the corresponding depth image. The second modification was made because initially a message would only be published on the /darknet_ros/bounding_boxes topic if objects were detected in the image. We needed the message to be published even if no objects were detected in order to enable easier synchronization later on between the bounding_boxes output and the corresponding depth image. A6 provides an example of output from /darknet_ros/bounding_boxes with the code modifications described above being applied. The output from darknet_ros onto the bounding_boxes topic can then be processed by the image_processor node.

The image_processor node synchronizes and combines information about the detected objects and information about image depths into a cohesive message. It is written in C++. Messages on the topics /darknet_ros/bounding_boxes and /kinect/hd/image_depth_rect are inputs for the node. The bounding_boxes messages provide information about the objects detected and their locations. The image_depth_rect messages provide information about depth values throughout the image. The messages are synchronized by matching or near matching timestamps. Then information from the synchronized messages is combined and output onto three topics. The three output topics for the node are /image_processor/all_objects, /image_processor/flag, and /image_processor/enemy_person. Messages that are published on the all_objects topic each object's probability, the boundaries of each object, the center point of each object, the each object's depth, and information about each object's identity. These messages also include the dimensions of the image, the full depth image (to be able to navigate around obstacles that are not explicitly identified by object recognition), and the image sequence numbers for both the colored image used for object recognition and the depth image. Messages on the all_objects topic are intended to be sent to the movement_processor node so that the information can be used in controlling robot motion (see A7 for sample /image_processor/all_objects output). Messages sent on the /image_processor/flag topic contain information about the probability, location boundaries, center point, depth, and identity of each flag object recognized by the image_processor. This information can enable the arm attached to the robot to be able to pick up the flag. The information is sent to the arm_processor node, which will be made by different students in a future semester. Messages sent on the /image_processor/enemy_person topic include information about the probability, location boundaries, center point, depth, and identity of each person and enemy robot recognized by the image_processor. This information enables the tennis ball launcher that will be attached to the robot to target the enemy robot, ensure no humans are near the robot, and then fire a tennis ball at the enemy robot. The information is intended for the launcher_control node that will be built by different students in a future semester. No sample output is provided from the image_processor and enemy_person outputs because we do not currently have access to the flag and the enemy robot, so we were not able to perform custom object recognition training with these objects. However, the all_objects output is currently being used by the motion node.

To allow the robot to move around the play area using a path based on its surroundings, we have created a ROS node that controls its motion. This node takes as input all_objects_info messages from the image_processor node. Using these messages, the robot either moves forward if the flag is in view and no obstacles are in the way or spins until both scenarios are true. To do this, the node subscribes to the all_objects topic and publishes motion commands to the /husky_velocity_controller/cmd_vel topic, to which the robot subscribes. If the flag is in view but not directly in front of the robot, the robot will pivot until the flag is straight ahead. If at any point an obstacle appears, the robot will turn, reevaluate for any other obstacles using the Kinect's lidar capabilities, and proceed if there are none. Due to hardware limitations at this point, the node cannot be accurately tested due to the latency between image capture and sending messages, so a demo has been created to test the motion commands for the robot. This works accurately, so we assume that with a more powerful GPU, the motion node will be able to run properly.

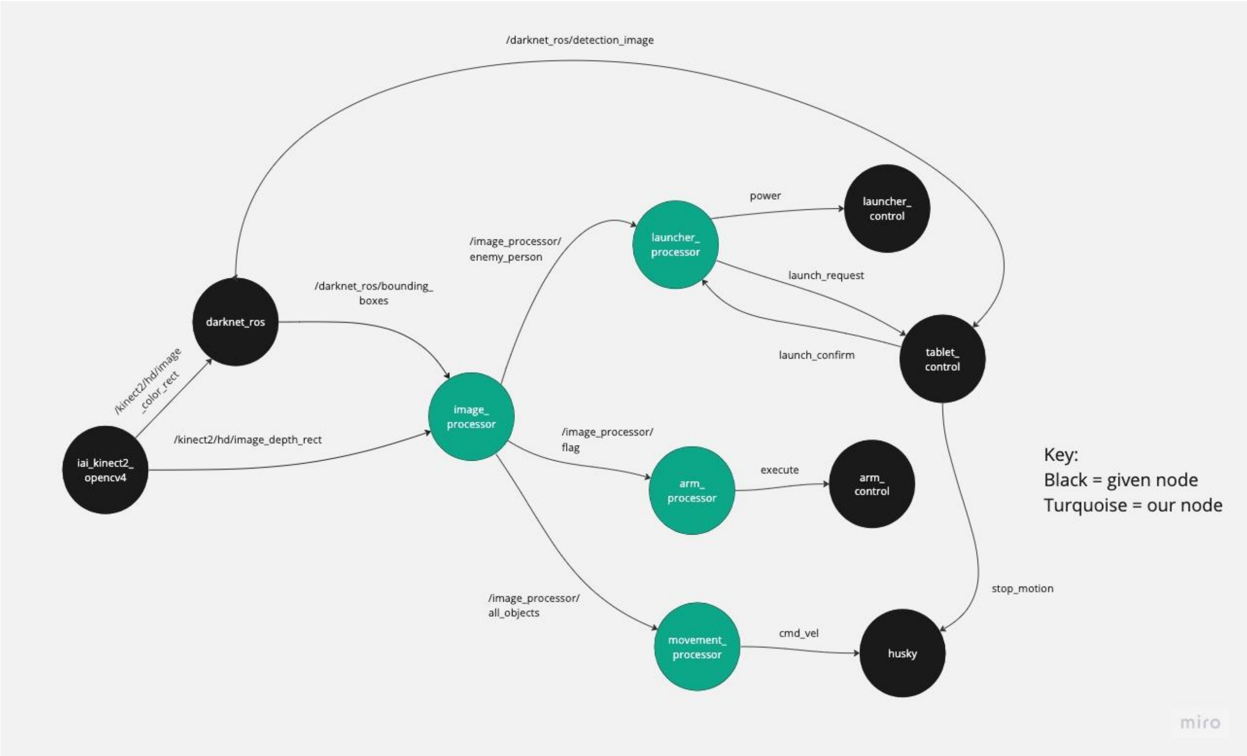
Conclusion

As two students wanting to enter the software field, we both understandably had little to no recent experience with robotics. Because this project hinged almost entirely on hardware, we had to work with both software and hardware components. As we worked further with the main components on our end of the project (the Kinect sensor and the Husky robot itself), we realized this was not going to be a simple task. With the help of our advisors and other team members, however, we were able to accomplish a large chunk of the project, leaving less work to be done for future semesters. As we pass this project off to the next team of students, they can build on what we have started and eventually create a fully functional capture the flag robot.

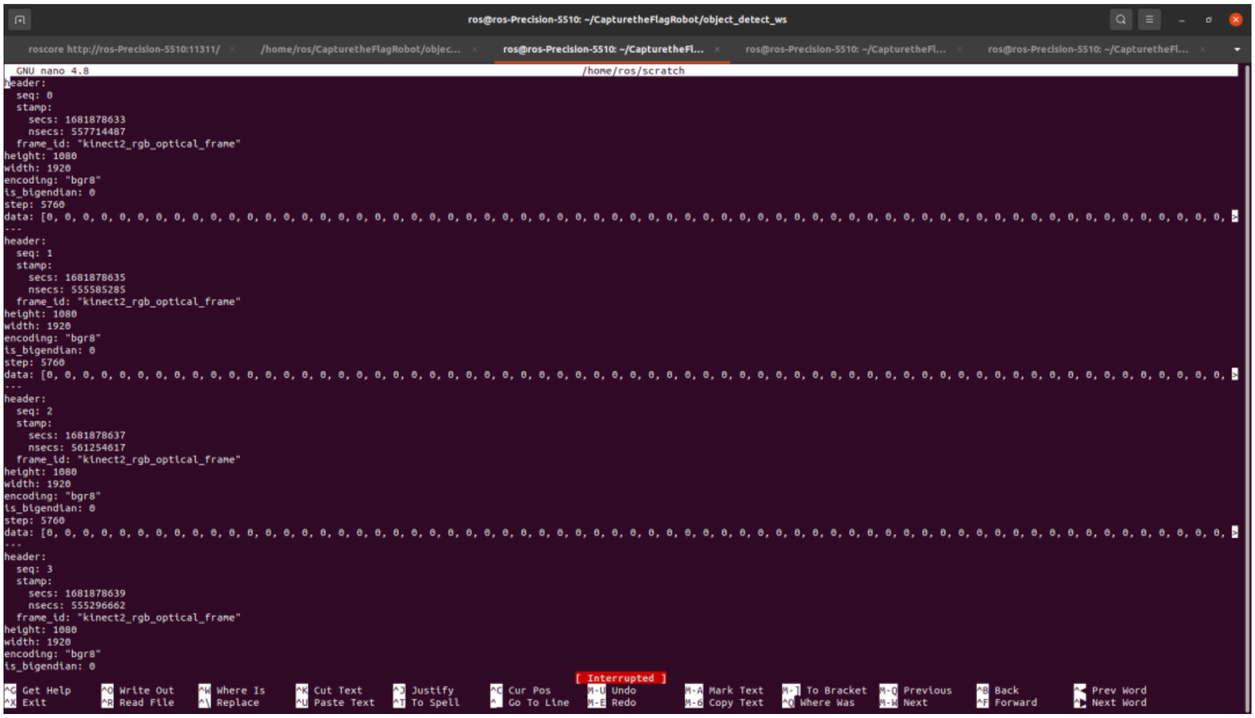
Appendix A

A1. Clearpath husky robot without top plate

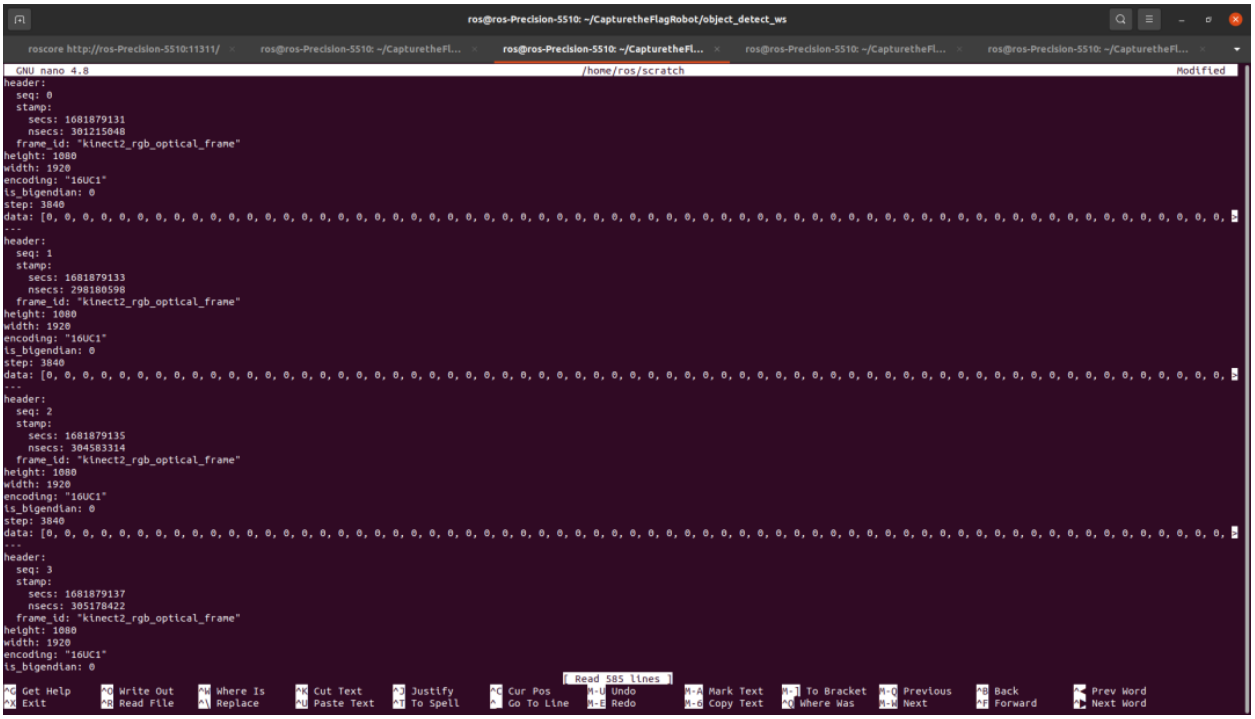
A2. Inputs/Outputs Diagram



A3. Sample output from the /kinect2/hd/image_color_rect topic

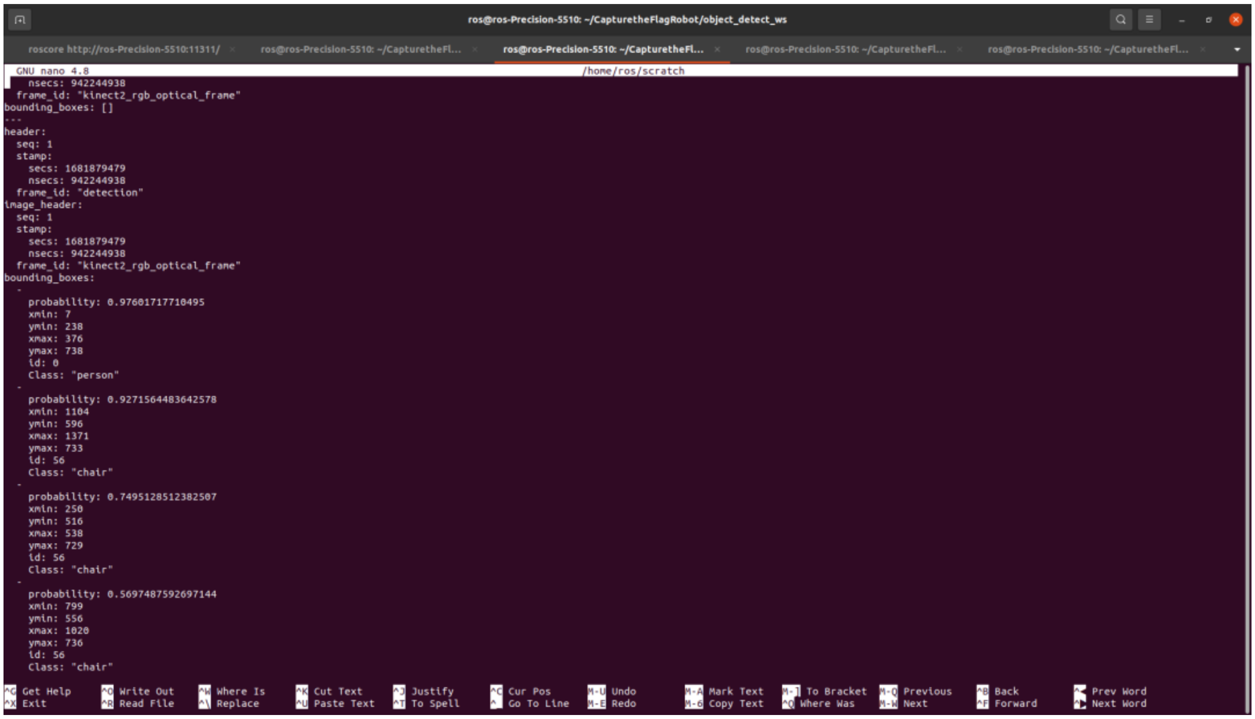


A4. Sample output from the /kinect2/hd/image_depth_rect topic

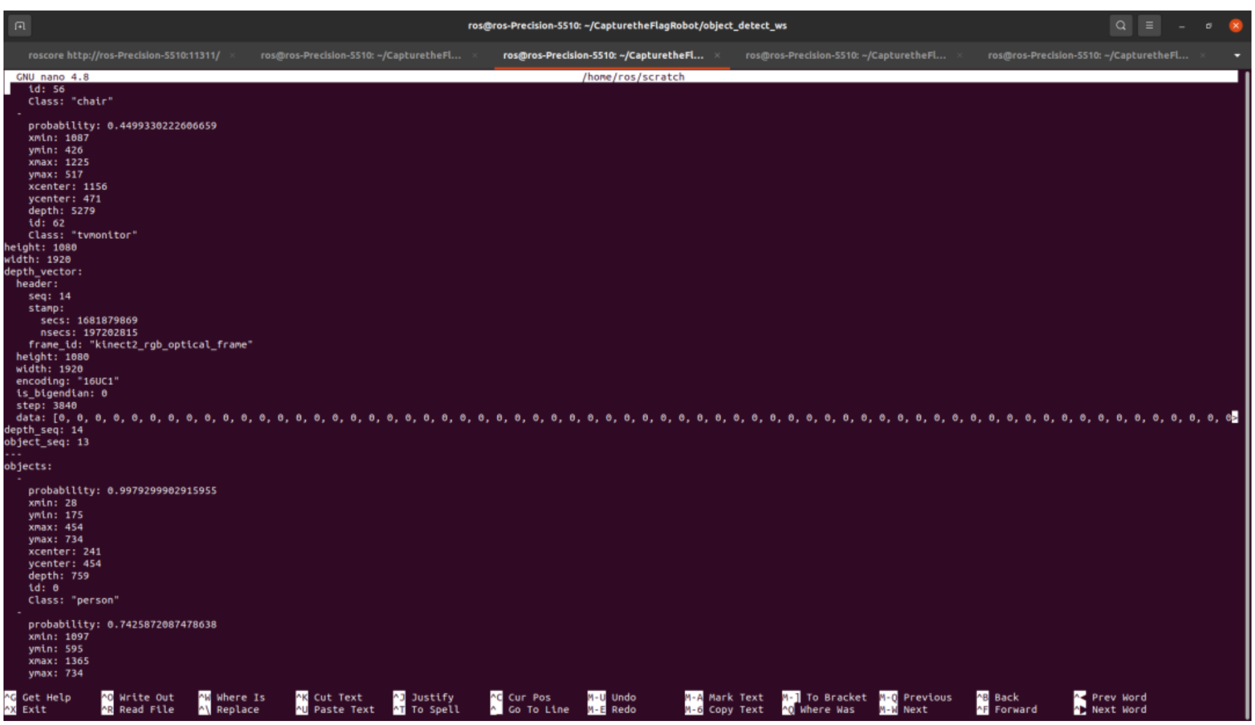


A5. Depth calibration setup

A6. Sample output from the /darknet_ros/bounding_boxes topic



A7. Sample output from the /image_processor/all_objects topic



Works Cited

Bjelonic, Marko. “leggedrobotics/darknet_ros.” *Github*, 30 Jun. 2021, https://github.com/leggedrobotics/darknet_ros.

Keita, Zoumana. “Yolo Object Detection Explained.” *DataCamp*, DataCamp, Inc, 28 Sept. 2022, https://www.datacamp.com/blog/yolo-object-detection-explained.

Olafenwa, Moses. “Official English Documentation for ImageAI!” *ImageAI*, 2022, https://imageai.readthedocs.io/en/latest/.

Paul, Shuvo. “iai_kinect2_opencv4/kinect2_bridge/.” *Github*, 28 Oct. 2020, https://github.com/paul-shuvo/iai_kinect2_opencv4/tree/master/kinect2_bridge.