OpenCV AI Competition 2021

Final Report

Team GrandPlay, ElsaBot Project

Introduction

This report documents the OpenCV AI Competition 2021 entry for Team GrandPlay. This was a single-person team consisting of Scott Horton. This project has involved creating a robot-based game platform, called ElsaBot, for hosting simple games targeted at small children for learning and entertainment. The primary goal of the robot and the games is to create a fun way to experience and explore technology with the hope that it leads to a future passion to learn more about science and technology. On a personal level, the goal of the project was to involve my 3-year-old granddaughter in the development process as the beginning of many ongoing projects as she becomes capable of understanding more aspects of technology.

Platform

The OAK-D camera is the key component of the custom ElsaBot mobile robot platform shown in figure 1.

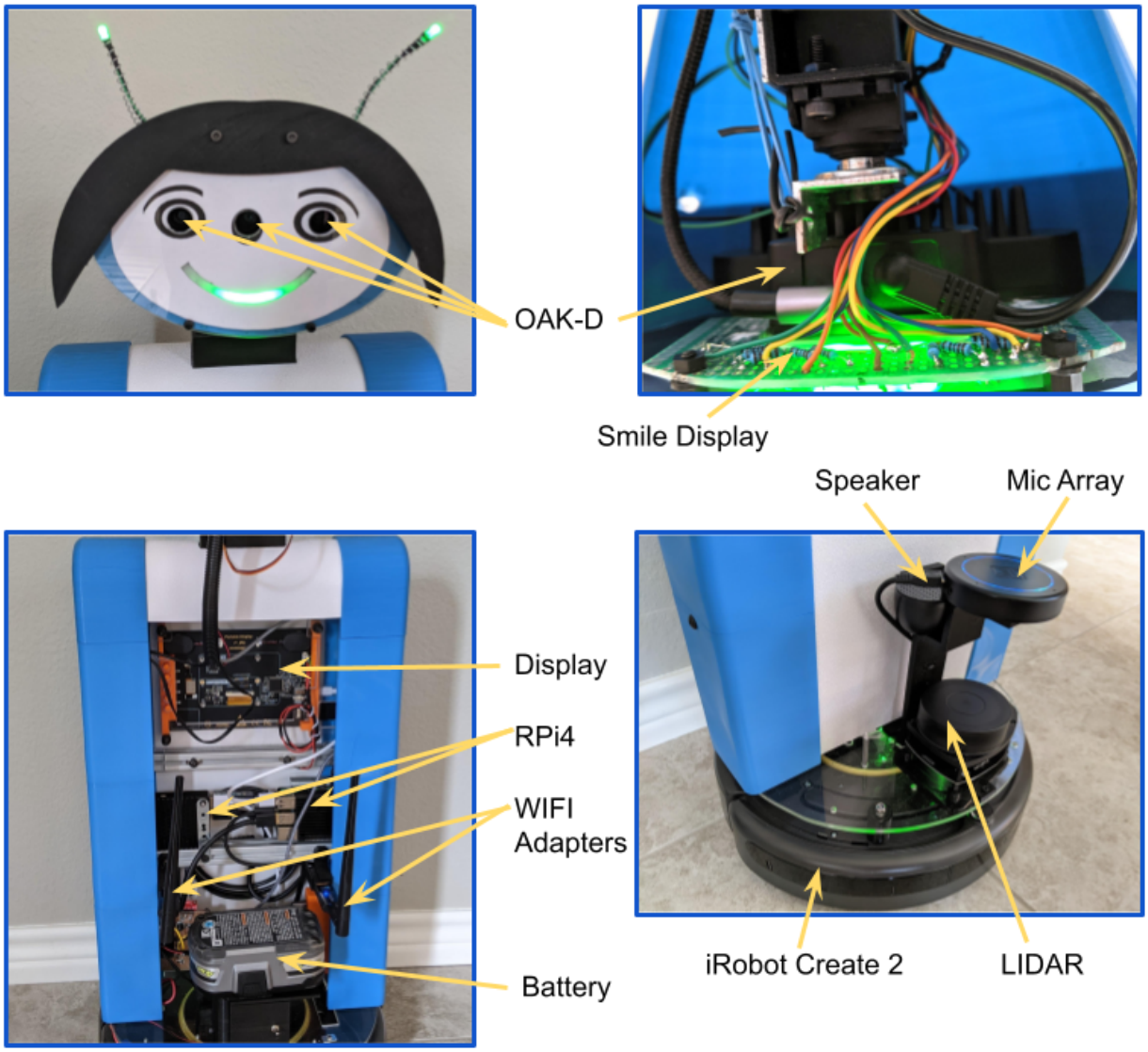

Figure 1

The robot structure is composed of an iRobot Create 2 base, a vertical frame, and a head unit mounted at the top of the frame. The head unit integrates the OAK-D camera such that the individual cameras are exposed via the eye and nose holes of the robot face. The head unit is mounted to the frame using a three-axis servo assembly. This allows the camera orientation to be changed to support tracking of a person in front of the robot or to allow scanning the room for a person. The goal for the appearance of the robot was to

make it a bit cartoonish and overall friendly to a child. Much of the enclosure for the robot was 3-D printed in sections.

The robot hardware includes:

- Raspberry Pi 4 single board computer (2)
- iRobot Create 2 base connected via USB-to-serial adapter
- Two USB WIFI adapters (one per RPi4)
- Display panel with touch control
- Slamtec RPLIDAR A1 Lidar
- Adafruit PCA9685 16-Channel Servo Driver (I2C interface)
- Custom LED smile display
- Head antennae (LEDs mounted on springs attached to head)
- ReSpeaker USB microphone array
- USB speaker
- USB hub
- Ryobi 19V Battery and DC-to-DC converters

Software Architecture

Both Raspberry PI computers run Xubuntu 20.04 as the OS with ROS 2 as the robot software framework. Most of the software developed or used for this project are ROS 2 software packages. ROS provides a framework that supports a modular approach to implementing a complex robot solution and also provides a rich and powerful base of robot services. These ROS packages run as nodes (typically separate processes) and communicate via the ROS framework. Figure 2 shows the key nodes used for this project.

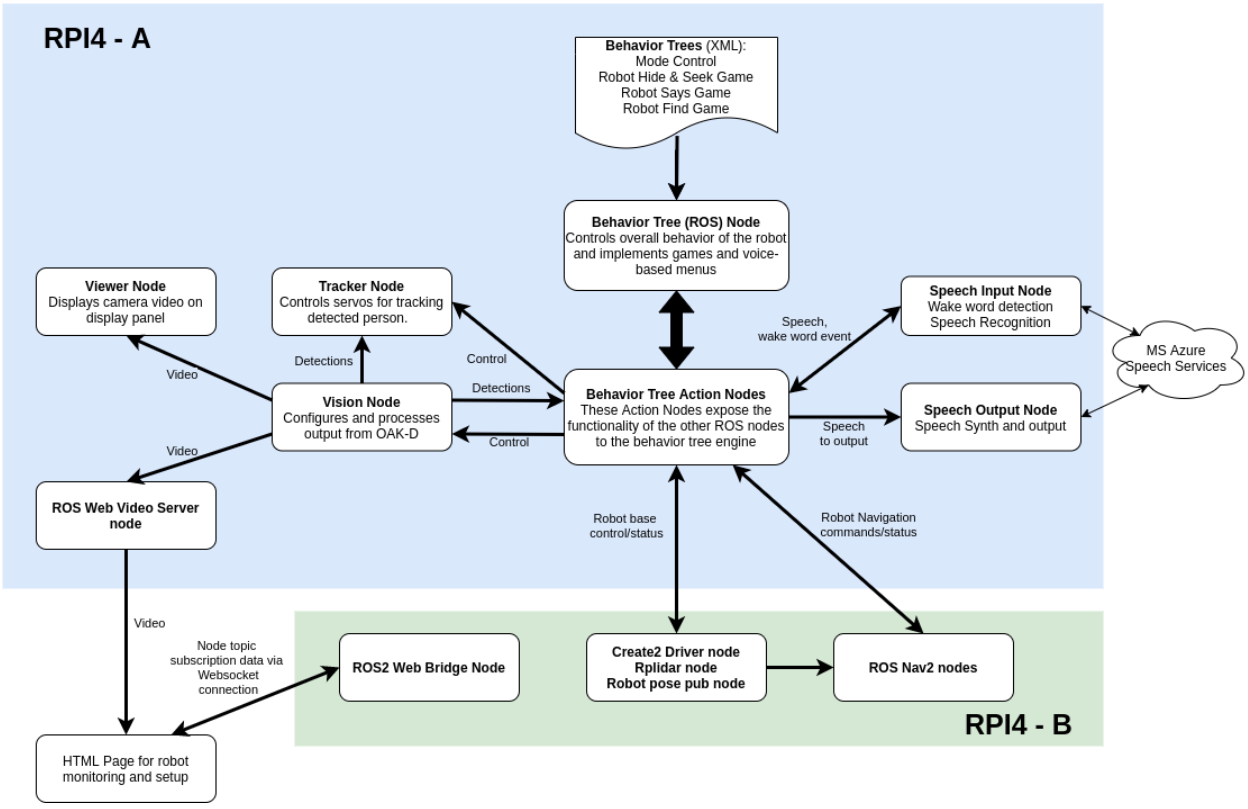

Figure 2 - Key ROS nodes

As shown, the nodes have been partitioned across the two Raspberry Pi computers (denoted A and B). Originally, only one computer was expected to be necessary for this project. However, it was found that the computing resources necessary for the robot base control and the navigation services of ROS Nav2 required more than 70% of the CPU and did not leave sufficient resources for nodes implementing the functionality unique to the robot (OAK-D control, speech processing, and top-level control implementing the games). Fortunately, since ROS supports a distributed computing model, it was relatively easy to add another RPi4 and partition the load across the two computers. It should be noted that

the CPU required for support of the OAK-D portion of the solution is small since most of the processing is done on the OAK-D device.

The nodes implemented on RPi-B will not be discussed in detail since they represent a very typical collection of ROS nodes required to control a robot base and provide navigation functionality. Most of the nodes running on RPi-A were developed as part of this project. These nodes includes:

- **Vision Node** - ROS Node for the configuration and processing of the output from the OAK-D. Also handles the post processing needed for the human pose detection neural network.
- **Tracker Node** - ROS Node that receives detection results from the Vision Node and controls servos for tracking a person by changing the camera (robot head) orientation.
- **Viewer Node** - Custom ROS Node for displaying the OAK-D video output on the front display.
- **ROS Web Video Server Node** - Off-the-shelf ROS node for publishing video feeds via HTML streaming to robot web UIs.
- **Speech Input Node** - ROS node that reads audio samples from the microphone and uses Microsoft Cognitive Speech Services to perform speech detection and wake word detection.
- **Speech Output Node** - ROS node that uses the Microsoft Cognitive Speech Services to convert text to speech and then output using the USB speaker.
- **Behavior Tree Node** - ROS node that uses the BehaviorTree.CPP framework to execute the custom behavior trees and actions that implement the ElsaBot functionality. These trees use custom behavior tree actions developed for this project.

The Behavior Trees listed in the diagram are XML files that describe a tree of nodes whose composition describes how the robot should respond to external inputs. A node can describe a sequence of other nodes that can react to the success/failure of a given node. This allows reactive behaviors to be implemented. The BehaviorTree.CPP site provides a very helpful tutorial for getting started with behavior trees.

The behavior trees developed for this project cover the overall robot menuing interface (primarily voice based) and the game implementations.

OAK-D Related Nodes

This section describes the OAK-D specific functionality and software implementation used for this project.

Vision Node

The Vision node is implemented as part of the ROS robot_head package that was created for this project. That package includes the Vision, Tracker, and Viewer nodes which are implemented in Python. (The initial plan was to implement these nodes in C++. However, it was found that using Python for the vision related processing was much easier and efficient from a development standpoint and was easily supported by the ROS framework.)

The Vision node is largely based on the various samples provided by the OpenCV/Luxonis DepthAI documentation. Figure 3 shows the pipeline configuration and overall processing.

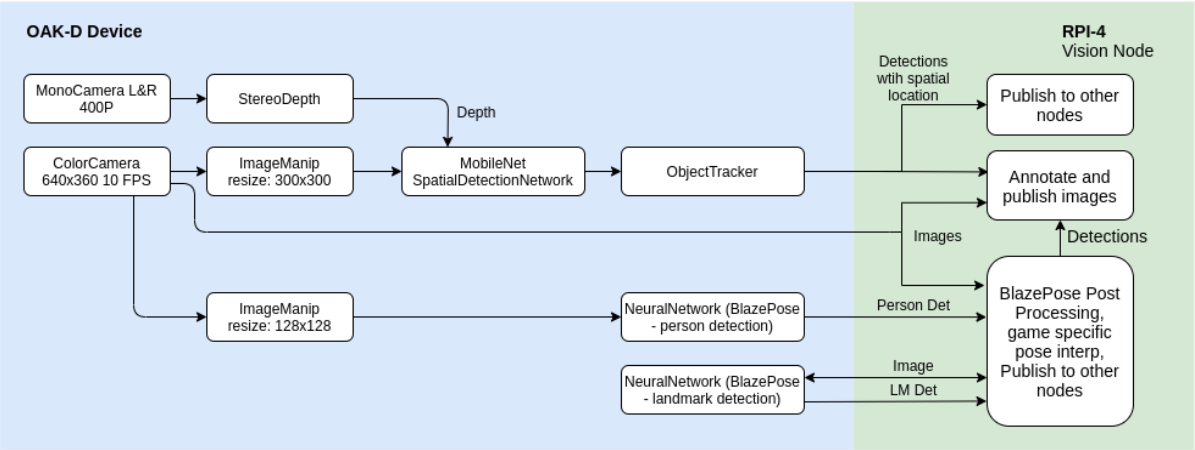

Figure 3

The top flow implements an object detection pipeline using MobileNet SSD that also provides spatial location information and object tracking. Support for Tiny Yolo V4 is also implemented by the vision node and can be switched in via a module variable. The MobileNet detector was selected as the default since it was found to provide better person detection when only the lower part of the body is in the field of view. The detections are published to other subscribing ROS nodes and the information corresponding to the object selected by the Tracker node (see below) is annotated over the image.

The lower flow implements the Google Mediapipe BlazePose human pose estimation pipeline (ported by Geaxgx to the OAK-D). This is a two stage detector that requires post processing on the host side. The final output of BlazePose neural network is fed into a

simple custom interpreter created for this application that detects several hand/arm poses used by the *Robot Says* game. The detection status is annotated on the image and published to other subscribing ROS nodes.

The annotated image is also published for display by the Viewer node and for streaming via HTTP to the robot webui running on a web browser.

Tracker Node

The Tracker node consumes the detection output of the Vision node and uses that information to control the camera/robot head servos to keep the upper body of the person in view. By default, the Tracker only processes person detections and will default to the closest person currently being tracked by the ObjectTracker pipeline node of the OAK-D. The Tracker node passes to the Vision node the ID of the person object that has been selected for tracking. This allows the Vision node to limit the image annotations and human pose detection for only that person.

The Tracker node publishes the servo positions to ROS. These positions correspond to robot joints that are described by the URDF of the robot. Using the published joint information, other nodes can transform the position information reported by the OAK-D for a detected object into coordinates relative to the map where the robot operates. This feature is specifically used by the *Robot Find* game.

The Tracker node also supports other functions such as:

- Scan left/right to find a person
- Turn to direction of detected voice/sound
- Turn Create 2 base to face the person being tracked
- Look-down gesture
- Head twist gesture
- Smile LED display control
- Antenna LED control

The Smile and Antenna control are implemented by this node out of convenience since they share the same hardware driver used to control the servos.

Other Key Nodes

The speech nodes play a key part of the robot experience by making the robot appear more lifelike. The Microsoft Cognitive Speech Services are used for this functionality via the Microsoft Cognitive Services Speech SDK. Speech recognition and synthesis is handled via

cloud requests using this API. Synthesized speech is cached locally to reduce cloud requests. The voice recognition requirements are very simple for this application since the target user is a child whose diction may not be clear. As such, most prompts only expect a 'yes/no' response.

The speech input node also implements a wake word detector as part of the SDK. The detector is trained for a specific wake phrase using a cloud-based tool of the MS speech services. The resulting neural net is used by the speech SDK to monitor the audio input stream for that phrase. Wake word detection events are published to subscribing nodes such as the Behavior Tree node.

Important ROS Nodes

ROS Nav2 provides a very powerful and robust set of navigation related nodes which allow the robot to navigate within the defined play area. After a map of the area is generated using ROS provided tools and the robot is given its initial location, the Nav2 stack of ROS can then be used to plan a path to a selected location on the map, and then control the robot base to execute that path while avoiding fixed and temporary obstacles. The *Robot Hide & Seek* game leverages this capability.

Source Code

The appendix of this document contains a description of the various packages that were developed and links to the repositories.

Behavior Tree Based Control

This project uses behavior trees to implement the high-level robot behaviors. While the BehaviorTree.CPP library provides many useful generic actions for tree control that are essential to creating a useful tree, it is also necessary to create tree action nodes that can be used to interface a tree with the available functionality of the robot platform. The following custom tree action nodes were implemented and are used by the ElsaBot games.

- Game related
    - Initialize a specific game
    - Get the data needed by other action nodes to implement a game step (ex. Pose to be expected, location to go to for searching)
- Detection related
    - Check for one of the specific human poses known by the robot
    - Check for a detected object
    - Control tracking/scanning
    - Get tracking/scan status
- Speech related
    - Output speech from text
    - Convert speech to text
    - Check for wake word
- Movement related
    - Navigate to a specified location in the play area
    - Spin in place a specified number of degrees
    - Get the location of the robot
    - Clear Nav2 local/global cost map
- Emotional expression related
    - Control smile display
    - Control antennae LED mode
    - Control head tilt expression
    - Control look-down expression

- Misc
    - Compare text strings
    - Get random selection
    - Save image from camera
    - Load game settings

Behavior tree nodes can have input and output ports which can be used for communicating between nodes. The ports specify values that are stored in a common ‘blackboard’ used by the tree. The ports allow parameters to be specified to a node such as the text to say for the case of the text to speech action node. Action nodes can also return data to the blackboard which can then be used by another node.

The Robot Games

Three games were developed for this project. The target audience for these games was a 3-year-old child. These games are:

- Robot Hide & Seek
- Robot Find (Number/Shapes)
- Robot Says

Robot Hide & Seek

This game was intended more for entertainment than for learning. For this game a list of pre-specified locations (ROS Nav map coordinates) are provided via a JSON file. Based upon the starting location of the robot, the robot determines the closest location to start the search by calculating a navigation path to each, and then selecting the shortest path. It then navigates to the first location, and then to each successive location until it finds a person.

The robot uses both the output of the MobileNet SSD object detector and the output of the Blazepose human pose detector to detect a person. The pose detector is also used since it was found that in some cases it better detects a person who is farther away. While the robot is moving and when it reaches each point, the camera is scanned left and right to increase the field of view for detection.

The voice synthesis capabilities of the robot are used to enhance the experience by starting the game with the typical count-to-ten and then 'ready or not, here I come' declaration. Additional voice messages are output during the search. When the robot sees a person during the search, it proclaims "I found you", and then offers to play again.

Robot Find (Numbers/Shapes)

The purpose of this game is to help teach numbers and shapes.

The setup for this game involves positioning nine cards on the floor each spaced approximately two feet apart. Each card has printed on it one of the numerals from 1 through 9. The robot is positioned a few feet away from and facing the grid. A setup step is then used by the parent where the robot calls out one of the numbers, the parent stands on the number, the parent raises their hand, and then the robot determines the location of the person relative to the room map. The robot records this coordinate, and repeats this process for each location. The location of the robot is also recorded for easy re-setup (assuming the cards are later repositioned on the floor in the same position).

For the actual game play, the robot asks the child to stand on a particular number. The robot then calculates the position of the child and determines if the child is within a specified tolerance of the known location. If the child is close enough, the robot gives verbal praise and flashes its antennae.

The position calculation needed for this game is relatively straightforward by using the spatial location of the detected person as determined by the depth capabilities of the OAK-D and using ROS to transform that camera-relative position to a map-relative position given the robot base position, robot orientation, and the current angles of the robot head servos used for camera tracking.

This game also supports placing cards with shapes over the numerals as a game variation. For this case the robot calls out the shape to stand on instead of a number. Other card sets could be used by updating the JSON game file to specify the mapping from the alternate set to the set of numerals.

Robot Says

The purpose of this game is to help teach left vs. right in a fun way.

For this game the robot asks the child to place their hand/arm in a particular pose. The following poses are supported by the game and were chosen to be simple for a child, and have a high likelihood of being accurately detected:

- Hand above head
- Hand on head
- Hand on shoulder
- Hand on hip
- Arm extended out
- Hand touching stomach
- Arm by side

The game includes four rounds. For each round the order of the poses is randomly selected and then the robot asks the child to make each pose. The rounds differ as follows:

- Round 1 - either hand can be used
- Round 2 - left hand only
- Round 3 - right hand only

- Round 4 - game randomly chooses left vs right

The Mediapipe Blazepose human pose recognition neural network is used for this game. The detection output is then used to calculate shoulder and elbow angles from which the above poses are determined. The object detector is used to determine the position of the person during the game to keep the camera pointed at the player.

The project video submitted with this report provides a demonstration of each game. The number of game steps or poses were reduced in some cases to reduce the length of the video.

Project Results

This project successfully applied AI vision and speech technologies in the creation of a fun and useful robot game platform. The capabilities and ease-of-use of the OAK-D via the DepthAI SDK made the AI vision aspects of this project very straightforward. The advanced neural-net based Microsoft speech technologies rounded out the AI part of the solution by providing human-like speech synthesis and very accurate speech recognition. By combining those AI technologies with the capabilities of ROS, BehaviorTree.CPP, and several other ROS packages, a very useful and flexible robot AI platform was developed which will be the basis for ongoing work and play.

The games developed for this project demonstrated how well AI technology can enhance interactive games. However, these games only scratch the surface of what can be done.

The project also highlighted the challenges of creating games for a 3-year child who at that age has an attention span that is probably best measured in milliseconds. The game design aspects proved to be more of a challenge than the technical aspects.

Future work will involve adding more games that are more interesting and/or engaging to a child. One possibility to help achieve this goal is to add support for multiple players to allow for teamwork or competition. For this, AI vision techniques could be used to uniquely detect each person.

From a personal perspective, the project achieved the goal of introducing robotics and AI at a very simple level to my granddaughter. However, the most important project result was the technology playtime that I experienced with my granddaughter. My hope is that this project will provide a lasting impression on her that sparks a future interest in science and technology.

Appendix

Project Source

All custom source code created for the project was partitioned into ROS packages. Those packages are provided in these repositories.

elsabot_bt - https://github.com/rshorton/elsabot_bt

> ROS 2 package providing multiple nodes that implement the ElsaBot behavior tree engine and game behavior trees. The engine and trees control the overall behavior of the robot by using the following packages.

robot_head - https://github.com/rshorton/robot_head

> ROS 2 package including nodes which control the ElsaBot robot head. These nodes include: Vision, Tracker, and Viewer. The Vision node depends upon the DepthAI SDK.

robot_head_interfaces - https://github.com/rshorton/robot_head_interfaces

> ROS 2 package defining the ROS messages used by the robot_head package.

speech_input_server - https://github.com/rshorton/speech_input_server

> ROS 2 package implementing speech recognition and wake word detection using the Microsoft Cognitive Speech Services via the Microsoft Cognitive Services Speech SDK.

speech_output_server - https://github.com/rshorton/speech_output_server

> ROS 2 package implementing speech synthesis using the Microsoft Cognitive Speech Services via the Microsoft Cognitive Services Speech SDK.

speech_action_interfaces - https://github.com/rshorton/speech_action_interfaces

> ROS 2 package defining the ROS messages and services used by the speech_input_server and the speech_output_server packages.

tracked_object_mapper - https://github.com/rshorton/tracked_object_mapper

> ROS 2 package implementing a helper for the Tracker node of the robot_head package. This node transforms the position of the tracked object published on topic /head/tracked to room map coordinates and then publishes to topic /tracked_object_map_position.

elsa_bot_b - https://github.com/rshorton/elsa_bot_b

ROS 2 package containing the overall robot launch file for running the nodes that run on RPi4 CPU B of the ElsaBot robot. These packages mostly include the robot driver for the Create 2 robot base and Nav2.

elsabot_game_data - https://github.com/rshorton/elsabot_game_data

Provides game data files used for the ElsaBot games.

Acknowledgement/Credits

This project depended upon other projects and technologies.

- OpenCV AI Kit and DepthAI by Luxonis
- Luxonis Team for their amazing product support
- ROS 2 and NAV 2 (Rolling release from 4/2021)
- Microsoft Azure Cognitive Speech Services
- BehaviorTree.CPP - https://www.behaviortree.dev
- Xubuntu
- ROS 2 packages
    - async_web_server_cpp - https://github.com/GT-RAIL/async_web_server_cpp
    - create_robot - https://github.com/AutonomyLab/create_robot
    - libcreate - https://github.com/AutonomyLab/libcreate
    - rplidar_ros2 - https://github.com/Slamtec/rplidar_sdk
    - robot_pose_publisher - https://github.com/GT-RAIL/robot_pose_publisher
    - ros2-web-bridge - https://github.com/RobotWebTools/ros2-web-bridge
    - web_video_server - https://github.com/RobotWebTools/web_video_server