*eyeRobot* Developer Documentation

By: Kevinphi Hoang, Rachel Palmer, Eddie Penta

**Overview:**

Artificial intelligence has always been a topic of interest in the software development world- however, the goal of this project is to create a robot that can assist in daily struggles. If eyeRobot can perform to the caliber envisioned, the result would be life-changing for blind individuals who cannot afford a service solution. eyeRobot will be able to travel from point A to point B without colliding with any obstacles in its path. In addition to this, the robot should be able to detect and recognize said obstacles and alert us of the varying objects. Currently in the general market, there are no real *service* robots available; the ones available to the public are designed only to perform “dirty” or repetitive tasks, such as ordinary household chores. An optimal eyeRobot is anticipated to be able to run its own market for artificial, automated assistive technology for the blind.

**Major Components:**

* *Hardware*
  + Robot with four wheels, powered by four DC motors
  + Raspberry Pi 3 Model B
  + Motor HAT
  + Power bank for Raspberry Pi 3
  + Xbox 360 Kinect
  + Battery pack with 6 AA batteries
  + Gyroscope
* *Software*
  + Raspbian Stretch OS
  + [Adafruit Motor HAT Python Library](https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library)
  + [Tensorflow Object Detection API](https://github.com/tensorflow/models/tree/master/research/object_detection)
  + [libfreenect](https://github.com/OpenKinect/libfreenect)
  + AI Client script for world building module
  + Keras Neural Network Library
  + OpenAI Gym Library with custom environment and Unity integration
  + Unity 5
  + Remote Locomotion Program
    - Client python script
    - Java server
    - Unity C# scripts

**Design:**

* Hardware Design
  + The robot chassis is based largely on the [Actobotics Junior Runt Rover](https://www.servocity.com/junior). We will use a Raspberry Pi 3 Model B and a [Motor HAT](https://www.adafruit.com/product/2348) to control the four motors. The wires to the motors, as well as the battery pack are connected to the Motor HAT, and the USB Power bank is connected to the Raspberry Pi
  + The Kinect will sit on the front of the robot. For prototyping purposes, the Kinect is powered through a wall socket, and the USB is connected to a laptop. This decision was made because more time would have been needed to make the Kinect fully portable.
  + The gyroscope will sit on top of the Kinect, connected via USB to the Raspberry Pi.
* Server Design
  + The server needs to be both lightweight and lean. As such we have decided that the robot will communicate with the server using UDP, while connected modules will communicate with the server using TCP.
  + The robot will receive instructions from the Update Motor Packet. This packet will tell the motor how much power to give the motor and in which direction.
  + The robot will send small Sensor Information packet to the server. The Sensor Information packet tells the server the current state of all the sensors/motors connected to the robot and will be sent whenever the Update Motor Packet is received.
  + Modules connected to the server can send instructions to any robot connected to the same server. This means multiple robots can be connected to one server. Robots are identified by a string name, which is given to the server when the Robot connects
  + Modules connected to the server can request Sensor Information packets from any robot connected to the same server.
* AI Design
  + OpenAI was used to train the robot on both a virtual and physical environment
  + Unity was used to create the virtual environment in OpenAI
  + Unity and OpenAI communicated using a simple TCP protocol to send common OpenAI commands such as; *reset(),* and *step(action)*. Unity would respond with the current state of the virtual robot that would be used as the *observation\_state* for the AI.
  + If a physical robot was being trained, then Unity will send the current action to the physical robot.
  + The OpenAI agent would be trained using reinforcement learning.
  + The scoring would depend on whichever Policy is currently being used, this is very useful to test out multiple different ways of scoring the robot.

**Future Design:**

* Hardware Design
  + Obtain a larger chassis to house the robot components. The chassis should be able to hold the two battery packs and the depth camera easily.
  + Upgrade the motors. Currently the robot moves slower than the average walking speed. To keep up with a human walking, the robot will need to move faster.
  + Change the depth camera to something smaller than a Kinect (provided that it will not dramatically increase the cost of the robot).
  + Provide tether for the robot to the human so that the human can follow the direction of the robot.
* Server Design
  + The raspberry pi and server should take advantage of UDP Broadcast to automatically find and connect to each other, rather than having to modify a config file
* AI Design
  + Add a module for object detection and voice assistance. The object detection should run alongside of the world builder so that the robot knows what the obstacles are and can identify them when the user asks to be taken to them.
  + Update the experience relay so that the robot does not need to be physically reset after an epoch. It should be able to start a new epoch from where the last one ended.
  + Spend more time training the robot so that it can learn a room unassisted. The goal of this is for the user to be able to run the robot overnight to learn its environment, and the visually impaired user can use it as a guide starting the next day.
  + Implement voice commands so that the user can tell the robot “take me to the refrigerator”. The robot will know where that is based on its training with the world builder and object detection modules combined.

**Project Goals:**

* Locomotion: Robot can move around an environment on wheels.
  + Raspberry Pi with Motor Shield will be able to direct the robot.
  + The Raspberry Pi will be controlled by server using UDP.
* World Building: Robot can map and navigate its environment.
  + The robot will use the depth camera to capture and navigate the world environment.
  + Use AI programming to create and navigate a virtual environment.

**Additional Goals:**

* Object Detection/Recognition: The robot can detect and recognize different objects.
  + Using the Kinect camera data, it can detect an object and navigate away from it.
  + Using the Tensorflow Object Detection API, robot can recognize different objects.
* Obstacle Alerts (Voice): The robot can alert the user of differing and obstructing objects.

**How it Works:**

***Client (Robot AI):*** Connects to Raspberry Pi through a UDP network connection and sends

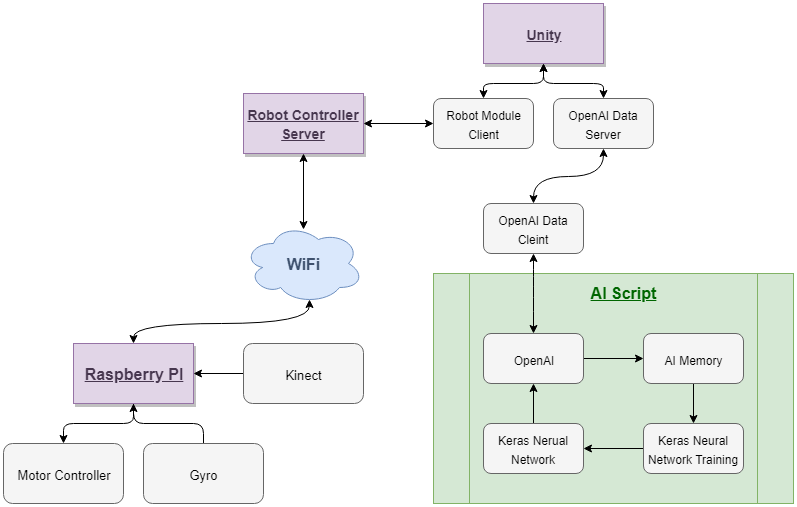
commands to the Raspberry Pi via the Robot Controller Server.

***Servers:***Receives commands from the Client and sends the data to the Raspberry Pi.

***Raspberry Pi:***Connects to the server using UDP and identifies itself. It will then

send receive commands from the server, and respond with new motor/sensor

information.



**External Funding:**

For the completion of the project, we required external funding for a few items:

1. ~20 AA Batteries **($10)**
2. Kinect 1.0 **($150)**
3. Gyroscope **($130)**

**Total:**  **$290**

This project was only possible thanks to Professor Wiseman’s financial help.

**Open Source Libraries:**

We used 9 different external, open libraries to develop eyeRobot:

1. Adafruit Motor HAT Python Library:

<https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library>

2. Tensorflow & Object Detection

<https://github.com/tensorflow/tensorflow>

<https://github.com/tensorflow/models/tree/master/research/object_detection>

3. OpenAI:

<https://github.com/openai/gym>

4. Keras:

<https://github.com/keras-team/keras>

5. OpenCL ML Training:

<https://github.com/plaidml/plaidml>

6. LibFreenect

<https://github.com/OpenKinect/libfreenect>

7. Java Networking:

<https://github.com/netty/netty>

8. Gyroscope Data:

<https://github.com/Knio/threespace>

9. Python Bytebuffer Library

<https://github.com/alon-sage/python-bytebuffer>