CS 4850 Section 02 Spring 2024

Robots Control and Vision

SP-1 Robots

By:
Blake Youngs
Heriberto Contreras
Brandon Tighe

Sharron Perry
April 14, 2024

Website:
https://github.com/Robots-Vision-and-Control

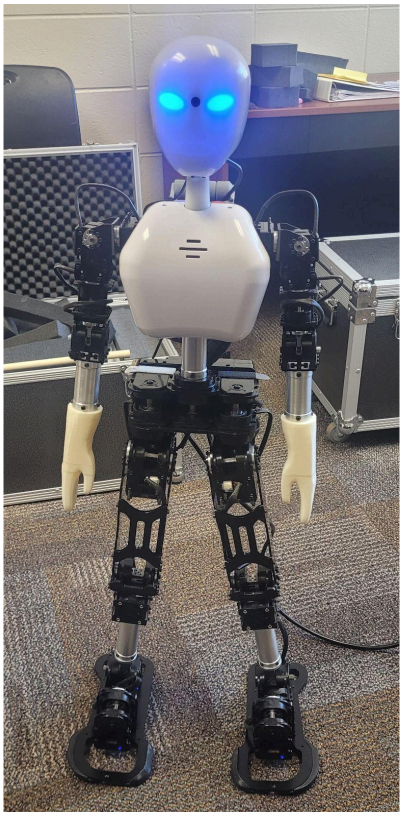
Github:
https://github.com/Robots-Vision-and-Control

Lines of Code:
628 LOC

Number of Components:
8

Table of Contents

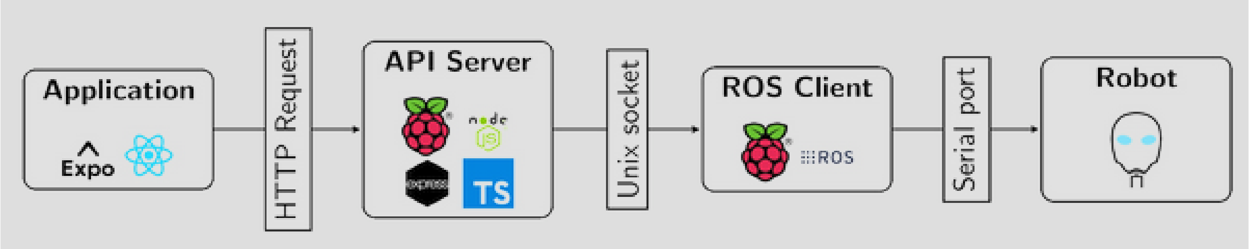
INTRODUCTION:



Kennesaw State University holds two UXA-90 model robots for student use. Previous groups have built entire projects around these robots ranging from creating a REST API for the robots to making the robots work with a VR headset. Our initial idea for a project was to allow the user to give voice commands to control the robot's actions. Alongside that feature, we wanted to grant the robot the ability to be able to point to an object via voice command. This feature would require the use of a machine vision model and an algorithm that would guide the hand to point to a desired object.

Previous Works

One of the key features that was added to Kennesaw's robots was the RESTFUL API. The API was developed by KSU alumni Derek Comella, Andrew Loveless, Sarah Thomas, and Jack Young. Alongside the API, this group developed a mobile application controller that would use the REST API to send commands to the robots. It is important to explain how The RESTFUL Robot's API works, as it will be used for the development of our project.



As seen above, the architecture is pretty simple. The application, or in our context, the Python code base, will asynchronously send HTTP calls to an API Server. Previous groups have used a simple Raspberry API with an Ubuntu server as their API server. We have chosen to do the same. The server communicates with the ROS Client which then provides the robots with motor or action commands.

Https calls explained

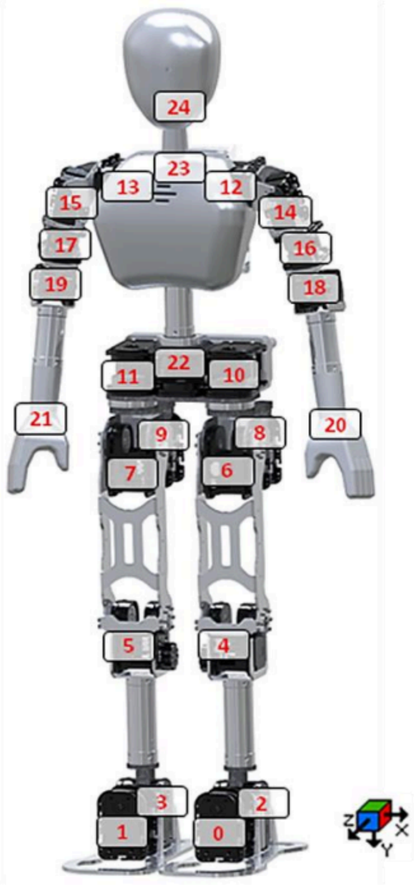
Since our project will be reliant on the RESTFUL robots API, it is only fair for us to explain how the HTTP calls interact with the robots. The HTTP strings are structured like this: http://123.123.123.123/12345/motion. The HTTP string contains an IP address (this IP address is collected from the API server), a port number, and a motion. The API supports pre-configure motions and individual motor control. Here are some examples of pre-configured motions:

http://123.123.123.123/50000/sit_down

This call would make the robot go into its sitting position. As mentioned before you can control individual motors.

http://123.123.123.123/50000/motor?id=12&position=50&torq=4

As shown above, this call has a bit more of a complex structure. After the port number, a motor is called using *motor?id=* then a number is used to specify the motor id. A position attribute configured next by *position= .* Here a position n number is called to indicate the position of that motor. Certain motors have different position ranges and defaults. The final part of the http string is the torq or *torq=* . Here the torque can be set between numbers 1-4. The number one indicates the strongest torque value and four indicates the weakest torque value. NOTE: For future reference, it is valuable to know that certain motors will not be able to use their full range because of the low torque being applied to a movement. It is recommended to use a stronger torq value than four.



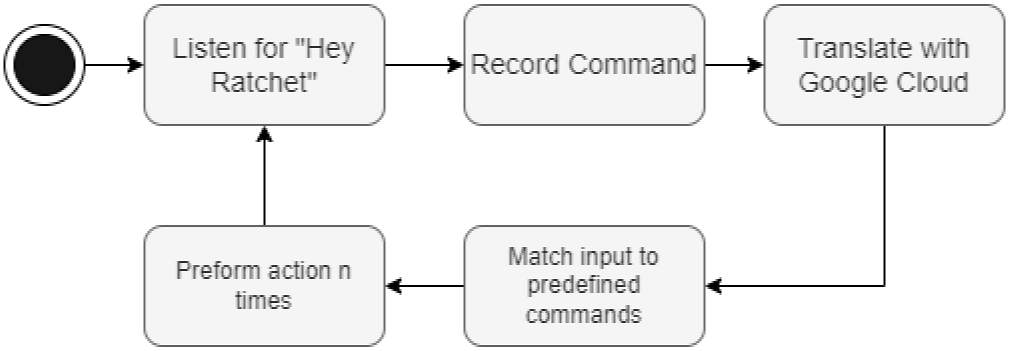
The diagram on the left-hand side is taken from the RESTFUL robots team's GitHub and it shows all of the possible motors and motor ID numbers.

Voice Controls and Commands

Overview:

Initially, when examining the UXA-90 robots we found them confusing to control and command. The robot has 25 controllable motors and a mix of pre-built motions that are a group of motor movements. A controller is provided with the robots, but only has 13 buttons and quickly starts to depend on pressing multiple buttons simultaneously to run and utilize a motor or motion. To address the issue we decided to take oral commands from a user and execute requests depending on what the user stated. Our goal is comparable to that of common voice assistants found integrated into various devices.

Our reason for going with a voice assistant approach is that it allows us to easily scale the possible commands for the robot without adding any complex controls. Additionally, it only requires a microphone to utilize which reduces the need to go through the process of updating a UI with the control or mapping a new combo on a controller.


Listen for "Hey Ratchet"
Record Command
Translate with Google Cloud
Match input to predefined commands
Preform action n times

Tools Utilized:

Picovoice Porcupine - gives an approach easy approach to hot word detection.
Picovoice Cobra - provides a way to detect if a user is speaking.
Pyaudio - allows access to utilize the microphone.
Wave - converts and outputs wave files for ease of use.
Google Cloud Speech-to-Text - utilized to convert the user’s command to a string
SpeechRecognition - Used as a backup for Google Cloud Speech-to-Text and is run locally on the host’s device.

Hot Word Detection:

We decided to go with hot word detection instead of constantly translating the incoming audio to text which is a waste of resources. So when looking into hot word detection we ultimately settled on Picovoice’s porcupine tool as it was easy to get working and cost nothing to use. When working with Porcupine we didn’t have to train a model for the phrase “Hey Ratchet”(The robot’s name) as they had tools for making a model super easily. After the hot word is detected a chime would play signifying to the user that the robot is listening. In all, hot word detection was something useful for reducing the use of resources and was easy to set up utilizing Porcupine.

Recording the Command:

When it comes to recording commands it is shown to be more complex and challenging than we initially anticipated. Timing is what showed to be the most troublesome part such as when to start recording, and when to stop recording. The primary goal is to minimize the amount of resources being utilized by keeping the length of the recorded audio to a minimum. So initially using another tool from Picovoice that allows for the detection of when a user is speaking.

When taking on the problem we would allow a maximum time after the hot word before it ignores recording the command to give the user time to think before speaking. Once the user started speaking it would start storing the incoming audio and record either until a set amount of time after the user finished speaking or a maximum allowed time is reached. By doing this it would stop the robot from recording any unnecessary audio and from cutting off mid-sentence for the user. Once the system is done recording the command it will play a different chime signifying it is done listening, and store the command as a local wave file to be utilized for translation.

Speech-to-Text Translation:

When it comes to translating the user’s command we had considered many alternatives but ultimately settled on Google Cloud’s Speech-to-Text API due to its usability, reliability, and supportability of multiple different languages. We built our system around English, but by utilizing their API we could easily expand the supported languages if we sought to. We additionally, used a simple SpeechRcognition tool as a backup if we were having issues with the internet connection which allowed the code to still function. In all, we ended up looking into an approach that was easily expandable and reliable.

Matching the Commands:

At this point, we have the user’s command as a string and a list of predetermined commands mapped to specific motions and motors. When it comes to matching the strings and making sure there could be some discrepancies and still match was important to make the experience as least frustrating as possible. When matching the two strings we make three different attempts at matching them.

Exact Matching:

In the first attempt at matching the strings, they are compared to see if they match identically. This is done to see if it can be caught right away before going onto more complex approaches that require more computations.

Word Matching:

In this approach, the user's command is broken down into just the words, and if all of the words in the command are found in a predetermined command they are considered to match and the command is selected. In all, this checks if all of the words in a command are called and returns the correct command if found. If not, the command goes onto the last attempt.

Fuzzy Matching:

In this last and final attempt, fuzzy matching is used to find a similar percentage between the two strings. If that value is above a specified threshold it will return the command found, and if it is not found no matching command would have been found. Overall, using 3 different approaches makes the system more reliable in finding the command which leads to less frustration.

The Final Process:

1. Listen for the hot word "Hey Ratchet"
2. Play a listening chime and wait for the speech
3. Record speech then the play-stopping chime
4. Transcribe the speech to text
5. Match the user command with a predetermined command
6. Pass the command to the robot for execution

In all, all these components come together to provide an easy-to-use and expandable approach to supplying commands and instructions to the robot instead of utilizing a limited controller that gets more confusing with each new action.

Image Detection and Classification

Overview:

A great functionality to grant these robots would be machine vision while also being able to incorporate tracking via voice commands. UXA-90 has a built-in camera in the middle of his forehead. The camera has a USB cord that comes out of the back of the head of the robot. For our purposes, we would connect the camera to a laptop that hosts the Python code base. This laptop has a GPU and will be utilized for live video/image processing.

Tools Utilized:

Ultralytics' YOLOv8 model - This machine vision model is great when it comes to live performance and it all hosts a lot of useful features for image detection, and segmentation.

OpenCV - This is a very popular machine vision library with a lot of useful tools for information collection and manipulation. In the context of our project, we will be using this library controller camera function that is not available in the YOLOv8 library.

YOLOv8:

Utralytics provide different sizes of YOLOv8 models. The sizes come in nano, small, medium, large, and extra-large. The larger the size, the greater the performance and computational resources. The robot's camera will detect an object and create a bounding box around the detected object and a confidence score. This machine vision model has an expansive amount of features that a user can configure. For this project, we will only be using the bounding box's information, ID numbers, and confidence scores. For more information about the YOLOv8 model and Ultralytics please visit https://docs.ultralytics.com/models/

Tracking/Auto Aim

Overview:

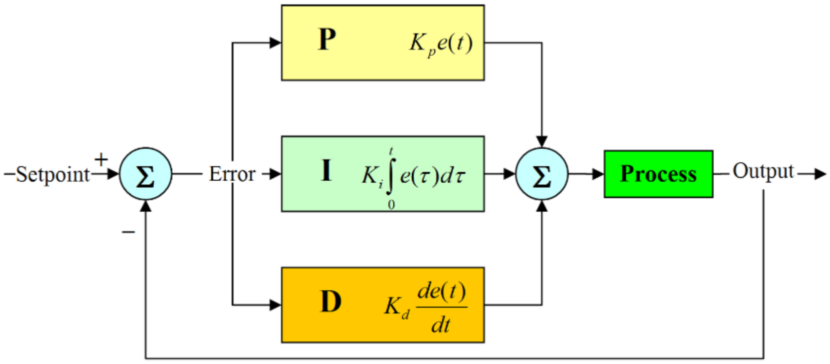
A great utilization of the YOLOv8 model is the ability to track and point at an object that is present in the camera's vision. Initially, our idea was to make the robot point to an object using its right arm. Compilation arose with that idea when we could not figure out why the robot's arm would not move based on its position numbers. The issue lays in the fact that our torque value was too low and the robot could not move any higher. This problem was solved too late into the semester, so it has been decided beforehand to make the pointing action of the robot move to a looking action. The tracking would be mapped out for the head motors so that the robot would be looking for tracking objects using its head.

Integration With YOLOv8:

The information that we will need from the machine vision model will be the coordinates of the center of a bounding box. The coordinate system is based on the pixels in a camera. For reference, the top left corner of a video feed will be the (0, 0) set of coordinates. Once we have the coordinates of our bounding box we must find a way to control the robot's camera so that the center of the camera overlaps with the center of the bounding box. To automate that action we will need a PID controller

PID Controller:

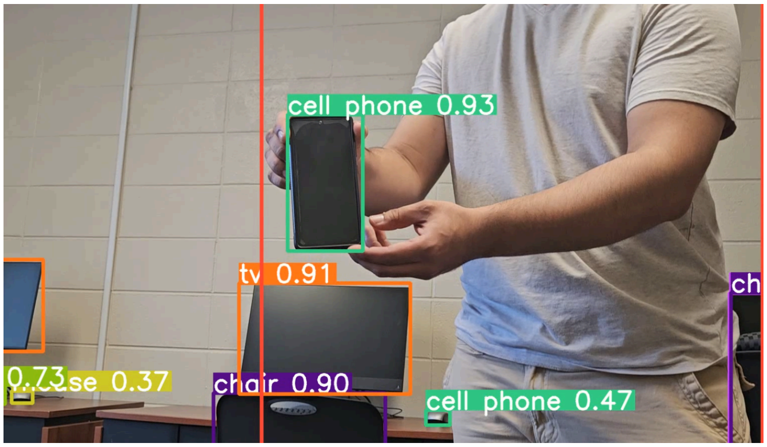
I will not be going over the extreme details of a PID controller but I can give out a brief overview.



A PID controller receives the output and a setpoint. It then calculates the error and runs the error through three different processes that all sum up to an output value that controls the system at play. In the context of our project, we calculate the horizontal and vertical error and try to minimize the error of the target bounding box and the center of the camera. This results in a controller that tracks the center of a target bounding box with great accuracy.

Integration with the Robot:

With the PID controller that reduces horizontal and vertical error, the only thing missing is the robot's head motor control integration. Both controllers give us the pixel count error between the middle of the camera and the middle of the target bounding box. The robot's head motor operational position has its position numbers. For the "up/down" motion of the head, the positional



numbers are 100 to 160 (100 being low and 160 being high). For our vertical controller to output a correct positional number instead of a pixel count, we took the ratio of pixels to position numbers (pixels: position numbers). This way the controller can output a positional number instead of a pixel number. This method also works for the horizontal controller. Together we get both controllers sending HTTP calls to both the neck and head motors with accurate positional numbers based on the target object.

Source Controller

For source control, we decide to utilize the tried-and-true git. Its versatility and widespread adoption make it perfect for easy-to-use source control. Git also makes it very easy to collaborate with multiple team members in a project.

Test Plan and Test Report

The implemented UXA-90s voice controls and commands can be tested exhaustively. At the moment the robot holds eleven voice-activated commands. Once the robot has been triggered to listen for a command:

Voice Commands	Http Calls
“Take # steps forward”	*http://123.123.123.123/50000/motion/walk_forward_short*
“Take # steps right”	*http://123.123.123.123/50000/motion/walk_right*
“Take # steps left”	*http://123.123.123.123/50000/motion/walk_left*
“Turn left”	*http://123.123.123.123/50000/motion/turn_left*
“Turn right”	*http://123.123.123.123/50000/motion/sit_right*
“Sit down”	*http://123.123.123.123/50000/motion/sit_down*
“Stand up”	*http://123.123.123.123/50000/motion/stand_up*
“Look up”	*http://123.123.123.123/50000/motor?id=23&position=x&torq=x*
“Look down”	*http://123.123.123.123/50000/motor?id=23&position=x&torq=x*

"Look right"	*http://123.123.123.123/50000/motor?id=24&position=x&torq=x*
"Look left"	*http://123.123.123.123/50000/motor?id=24&position=x&torq=x*

As shown above, the voice commands are paired with their respective http calls. Testing the commands is as easy as calling the voice commands and seeing if the proper http call where sent to the API Server.

Once the robot's vision is activated, the robot can track certain items listed in the pre-trained yolov8 documentation. Every object has an ID associated with it. Once the robot is tracking the desired object, the robot can be tested by simply moving the tracked object around.

Challenges and Risks

- Poor documentation of the RESTFUL API
  - After spending time on this project, the documentation for the API could be more verbose. We spotted gaps in command documentation and single motor control variables.
- One robot in poor condition
  - Kennesaw State University has two UXA-90 robots named Ratchet and Clank. One of the robots has an issue with the USB port used to control the robots via the API server. That rendered us with only one robot to work with.
- Now two robots in poor condition
  - Nearing the end of the project, one of our team members, who shall remain unnamed, had an incident where the cable that connects the robot to the API server ripped off the USB port. All that was left was a mangled port that was non-functional. That left us with no functional robot for us to make progress with our integration of systems.

Conclusion

The UXA-90 robots have been worked on and utilized at KSU for several years and have needed a better method of controlling the robot and giving it better approaches for interacting with its environment but primarily understanding the environment around itself. To achieve the first goal of allowing the addition of commands to be user-friendly, and easy to understand with the voice command system. By using something similar to a voice assistant users would be familiar with that kind of technology and would allow future teams to easily expand upon the commands without much difficulty. Moreover, machine vision is an ever-expanding field and we wanted to incorporate a model for object detection. A fun functionality that incorporates machine vision and robots is the ability to look at a detected object.